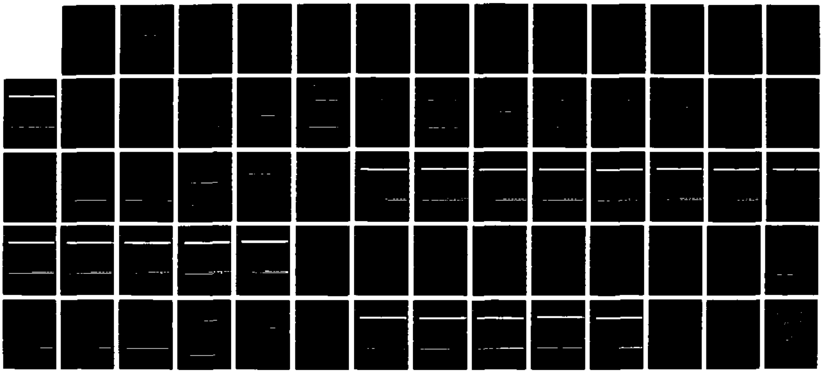
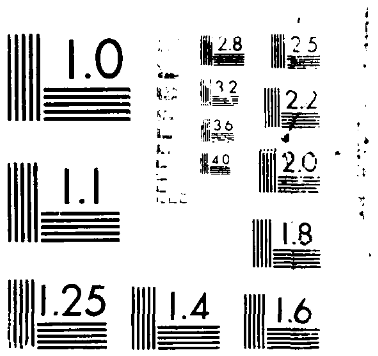
AD-A195 063   AN IRUS INTERFACE(U) BBN LABS INC CAMBRIDGE MA   1/1
K KOILE ET AL. MAY 86 BBN-6261 N00014-85-C-0016

UNCLASSIFIED   F/G 12/6   NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

DTIC FILE COPY (2)

BBN Laboratories Incorporated

A Subsidiary of Bolt Beranek and Newman Inc.

Report No. 6261

An IRUS Interface

Kimberle Koile and Edward Walker

May 1986

Prepared for:
Defense Advanced Research Projects Agency

[illegible]

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER BBN Report No. 6261	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) An IRUS Interface		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER BBN Report No. 6261
7. AUTHOR(s) Kimberle Koile and Edward Walker		8. CONTRACT OR GRANT NUMBER(s) N0014-85-C-0016
9. PERFORMING ORGANIZATION NAME AND ADDRESS BBN Laboratories Inc. 10 Moulton Street Cambridge, MA 02238		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE May 1986
		13. NUMBER OF PAGES 63
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Natural language interface, IRUS, Strategic Computing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes an interface to IRUS, a natural language system currently running on a Symbolics Lisp machine. The interface provides multiple pane displays and mouse and keyboard input and can be tailored to the needs of a particular application by building upon the generic capabilities described and illustrated in this report. The current applications for this interface include FRESH, an expert system developed as part of the Strategic Computing Program, and WQL, a window-oriented relational database query language.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Report No. 6261

An IRUS Interface[1]

Kimberle Koile and Edward Walker

May 1986

Prepared by:

Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, Massachusetts 02238

Prepared for:

Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

[1]The work presented here was supported under DARPA contract #N00014-85-C-0016. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government.

TABLE OF CONTENTS

LIST OF FIGURES

An IRUS Interface

1 Introduction

This report describes an interface to IRUS, a natural language system [2] currently running on a Symbolics Lisp Machine. Applications with which IRUS has been integrated include FRESH, an expert system developed as part of DARPA'S Strategic Computing Program, and WQL, a window-oriented relational database query language. For a description of how IRUS has been integrated with FRESH see [1]. For a description of WQL, see [3].

2 Types of IRUS Interface Configurations

Applications of IRUS require that the interface adopt two different types of configurations. 1) an end-user[2] configuration that supports easy and flexible input, editing, and processing of typed or mouse-selected queries and commands; 2) developer configurations for natural language experts who are developing, testing, debugging, or updating IRUS. The available IRUS interface configurations use the Symbolics Lisp Machine window system to provide multiple pane displays and mouse and keyboard input. The interface can be developed from the generic capabilities described and illustrated below and tailored to the needs of a particular application.

3 Components of the IRUS Interface

Three basic panes form the building blocks of the interface. The relative sizes and visibility of panes, the general appearance of the screen, the contents of menus, etc., are readily modifiable to suit the particular end-user community or application environment of IRUS.

1. An **Editing pane**, which has Zmacs[3] entry and editing capabilities and mouse selectable commands, supports entering and processing queries; queries can be typed, selected by mouse, or entered from a file of stored queries.

[2]The term "end-user" refers to someone who is not involved with IRUS system development.

[3]Zmacs is the editor provided on the Symbolics Lisp Machine.

2. A **History pane** maintains a list of successive queries and, where appropriate, responses. Queries in the History pane are mouse selectable, and pop-up menus permit selection of further actions on the selected item.

3. An **Interaction pane** provides detailed information concerning the results of each stage of IRUS processing. Entries in the Interaction pane are also mouse selectable, and a variety of pop-up menus permit selection of further actions on selected items.

A typical end-user's interface for IRUS would consist of an editing pane with a subset of the available commands for query entry, editing, and processing. A typical developer's interface would contain a configuration with all three types of panes to provide easy entry of test queries, checking of responses for related queries, and development and testing of IRUS internal structures such as the Meaning Representation Language (MRL), the Extended Relational Language (ERL), and the target language code generator.

IRUS acquisition configurations for adding new words and new domain interpretation rules to IRUS are provided for the developer and contain simple acquisition command menus and prompt and respond dialogs. Tools for domain model acquisition are provided by the knowledge representation system used by IRUS.

Depending on the nature of the application environment and of the items to be entered, these facilities might be adapted to provide system maintainers or even end-users with a means of expanding, maintaining, or updating the system.

3.1 Editing Pane

The Editing pane is used in the IRUS interface as a query input editor. The full Zmacs text editing facility of the Symbolics machine is available to the user for composing and editing queries. In addition to efficient text entry and editing, this editor supports macros, abbreviations, and other sophisticated editing features, which can be used to provide an efficient and customizable query entry system. The Editing pane also provides for aggregating queries and for writing queries into or reading queries from a file.

Mouse selectable **Edit**, **Process**, **Interrupt**, and **Other** commands have been provided.[4] These commands can also be invoked with keystrokes. A help facility,

[4]Unless otherwise stated, it is assumed that commands are selected by placing the mouse cursor over the command name and clicking the left mouse button.

available by selecting the **Help** command or by pressing the **Help** key, has been provided. Help messages, as well as specialized query composition aids, can be added in response to user evaluation. For example, a command menu of simple editing commands might be provided to aid users who do not know which keystrokes are used for Zmacs editing functions, or a brief explanation of the types of acceptable queries might be incorporated into the help feature.

The developer's Editing pane can be used not only for editing English queries, but also for editing the Lisp expressions produced by intermediate stages of IRUS processing. Full Zmacs text and Lisp editing functions are available, depending on the editing mode, which is either set automatically by the system or selected by the user. In addition to allowing a choice of editing modes, the developer's Editing pane contains commands that allow selective break points in the processing of a query. The developer may choose to halt processing at any of several stages of IRUS processing. This feature makes all stages of processing available for examination and editing, and it is especially useful for debugging and experimental development of the individual components of IRUS.

3.2 History Pane

The scrollable History pane accumulates queries and responses. Queries in this window are selectable for editing, reprocessing, or for requesting a more detailed description in the Interaction pane. This reentry feature can be used to provide a convenient standard form for a sequence of similar queries.

3.3 Interaction Pane

The Interaction pane provides a mechanism for detailed display of the queries and Lisp expressions that result from stages of IRUS processing. Items in this pane may be selected for more detailed examination or for further editing. For example, the results of parsing and semantic interpretation may be displayed and edited, then submitted for further processing to test alternative versions of initial query processing

4 Example Interface Configurations

The example screens in Figures 1 and 2 show typical end-user and developer configurations

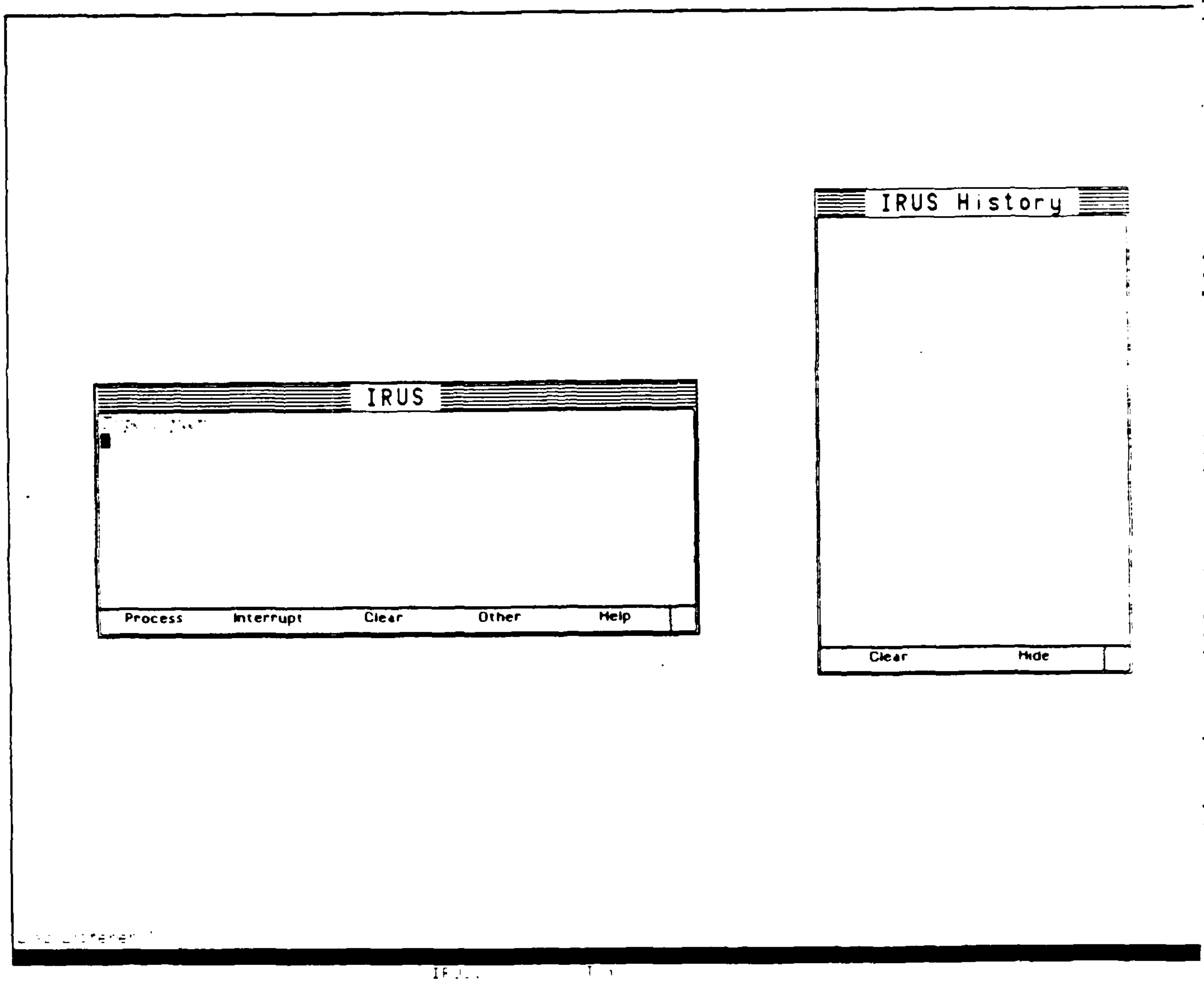


FIG. 1. TYPICAL END-USER CONFIGURATION

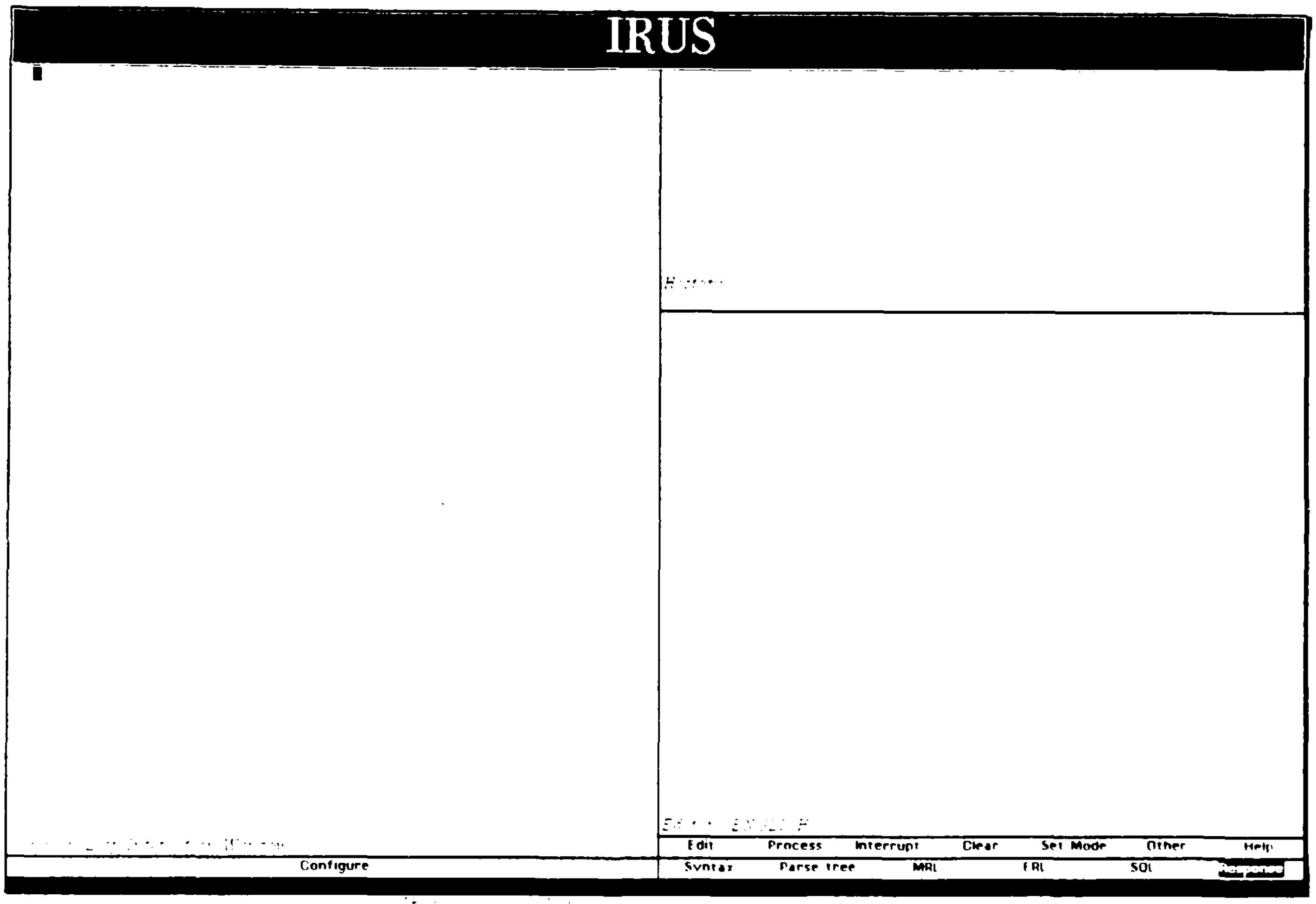


FIG 2 TYPICAL DEVELOPER CONFIGURATION

The end-user's version of the interface contains an Editing pane and a History pane in separate popup-up windows (see Figure 1). The pop-up History pane is optional and only appears when the user requests it (by selecting the **History** command from the pop-up menu that appears when the **Other** command is selected). The location on the screen and size of the pop-up windows are arbitrary and could be modified to reflect the availablity of screen space or the needs of the user. For example, a window could be popped-up in a designated area of the screen and remain available for a number of related queries. The window can be moved easily by using the mouse: the end-user moves the mouse cursor to the top portion of the pop-up window, holds down a mouse button, moves the mouse until the window is in the desired location, then releases the mouse button. The window also can be resized in a similar manner: the end-user places the mouse cursor over the lower right corner rather than the top of the window. An Interaction pane could be provided in either of the existing pop-up windows, in a fixed place on the underlying screen, or in a separate pop-up window. In this example, it is assumed that the response to the query will be handled by the application program using IRUS,and no Interaction pane has been provided.

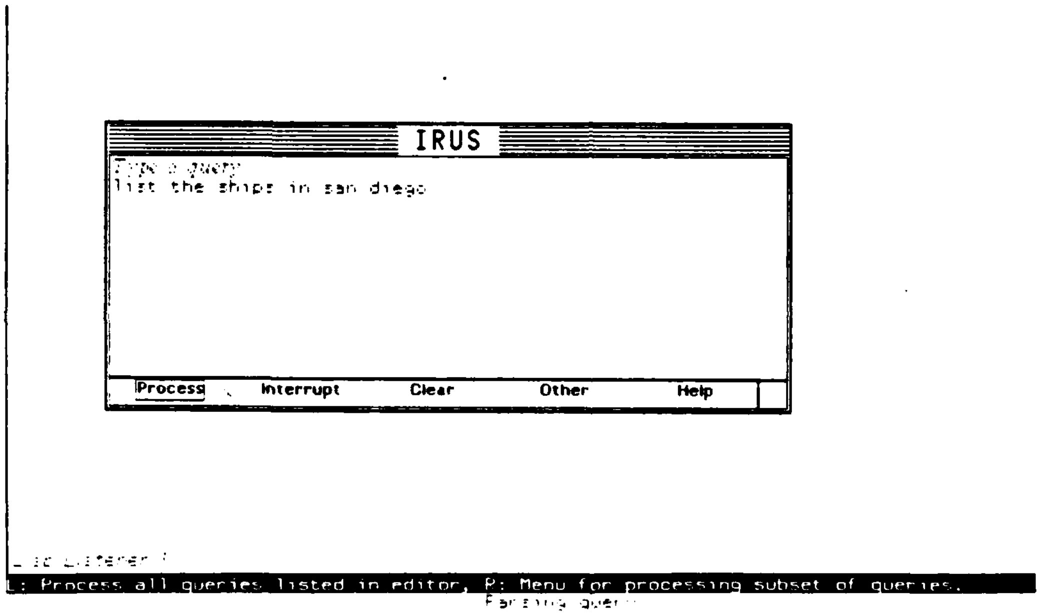
The developer's configuration illustrated here occupies the full screen and contains Editing, History, and Interaction panes (see Figure 2). An added command menu is provided to allow the developer to interrupt processing of a query at intermediate stages. The Interaction pane is used for detailed display of queries and intermediate processing steps. This pane is also a Common Lisp window, and the developer can type Lisp expressions in the pane and have them executed as if typing in a Symbolics provided Common Lisp window.

5 Interaction with IRUS

The following scenarios illustrate typical interactions with IRUS.

5.1 End–User Interaction

In the first scenario, an end–user has selected the IRUS pop–up window (from a system menu, for example). A query is typed into the Editing pane using the keyboard and processed by pressing the **End** key or selecting the **Process** command at the bottom of the window. As processing takes place, the current stage of processing is printed in the system status line at the bottom of the full Symbolics screen in order to provide user feedback. (See Figure 3.)



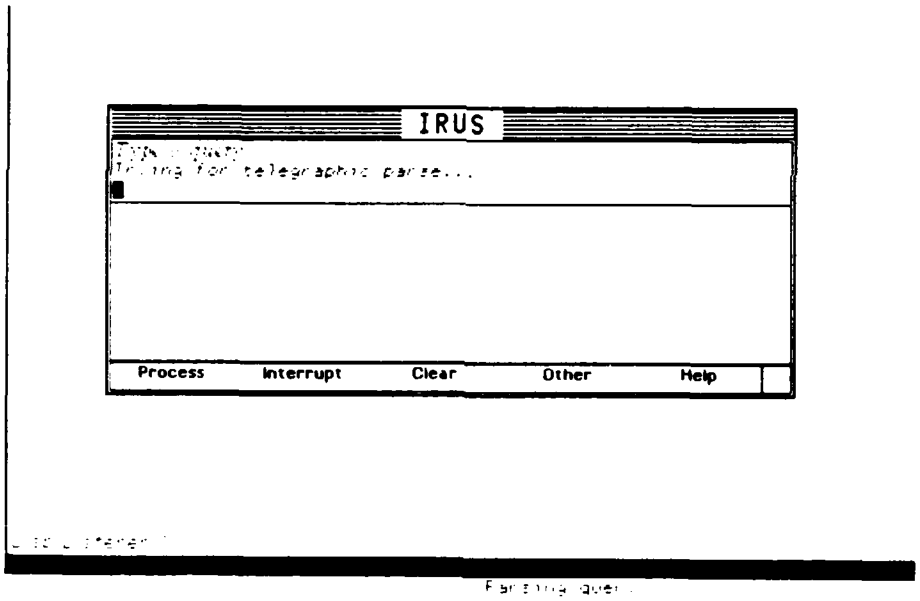
```
list the ships in san diego
```

```
L: Process all queries listed in editor, R: Menu for processing subset of queries.
Parsing query
```

FIG. 3. QUERY TYPED AND **PROCESS** COMMAND SELECTED

Upon the completion of IRUS processing, the IRUS pop-up window disappears, a query object is returned, and the application program (e.g., FRESH) takes over command processing and response handling. These query objects contain the query string, the results of stages of IRUS processing, and input data needed for the application program. Should the user want to interrupt query processing, the **Interrupt** command can be selected or the **Abort** key hit.

Messages to the end-user are shown in a window that scrolls down over the editor pane. If, for example, IRUS processing does not succeed, IRUS attempts a telegraphic parse and prints "Trying for telegraphic parse" in the scroll down window (see Figure 4). Typing a space causes the scroll down window to disappear. If the telegraphic parse fails, IRUS prints a failure message, leaving the user in input mode, ready to enter another query, press the help key, etc. The message facility can be used to notify users of failure and provide users with help messages for correcting errors of query form or content.

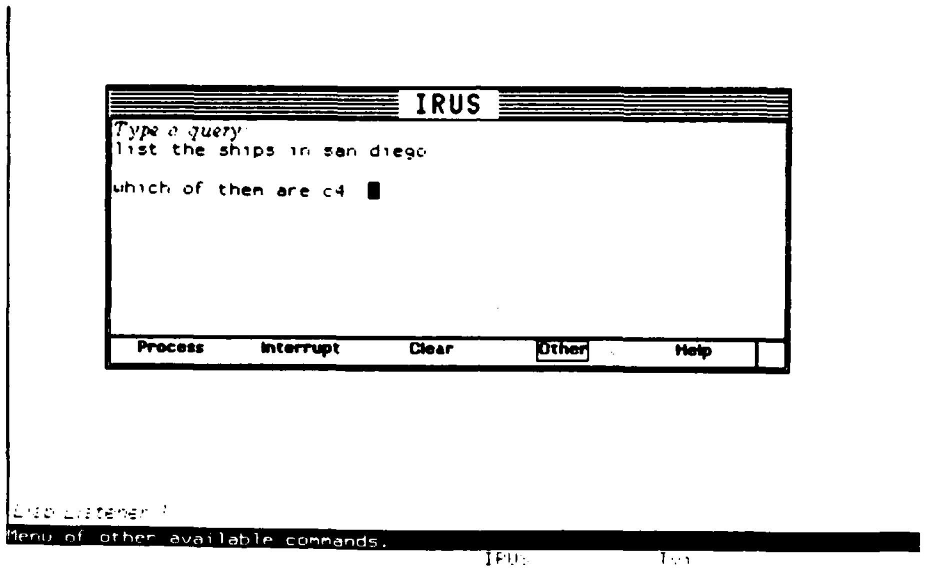


```
Type a query
Trying for telegraphic parse...
```

```
Parsing query
```

FIG 4. MESSAGE TO END-USER

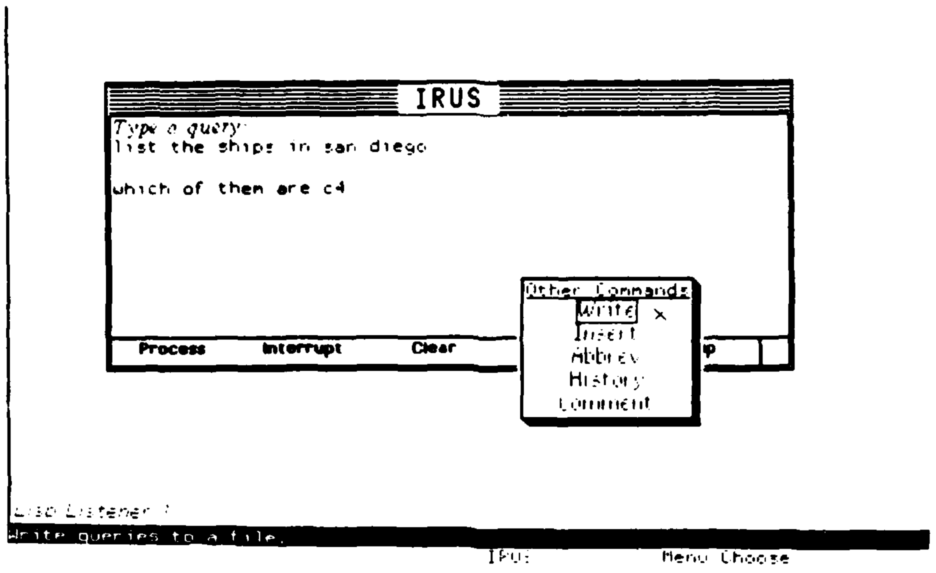
Queries may be aggregated in the Editing pane by separating them with 2 carriage returns. All queries will be processed in sequence after the **End** key is pressed or the **Process** command is selected. Queries can be written from the Editing pane into a file or retrieved from a file and inserted into the Editing pane by selecting the **Other** command, which pops up a menu, then selecting **Write** or **Insert**, respectively (see Figures 5 through 11). This facility is useful for creating a set of queries that are processed on a regular basis.



```
Type a query:
list the ships in san diego

which of then are c4
```

```
Lisp Listener 7
Menu of other available commands.
                                     IRUS          Tyi
```

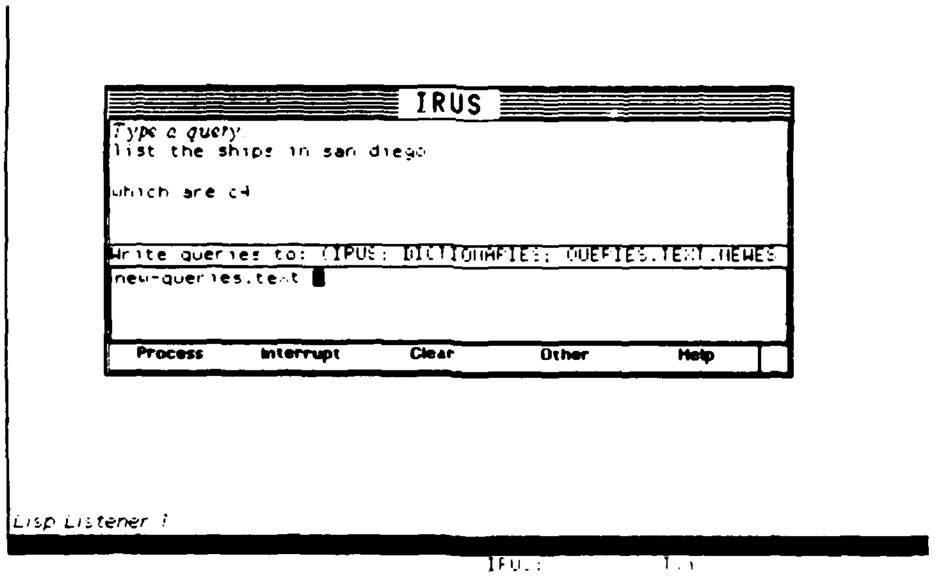
FIG. 5. SELECT OTHER COMMAND



```
Type a query:
list the ships in san diego

which of then are c4
```

```
Lisp Listener 1
Write queries to a file.
IRUS                    Menu Choose
```

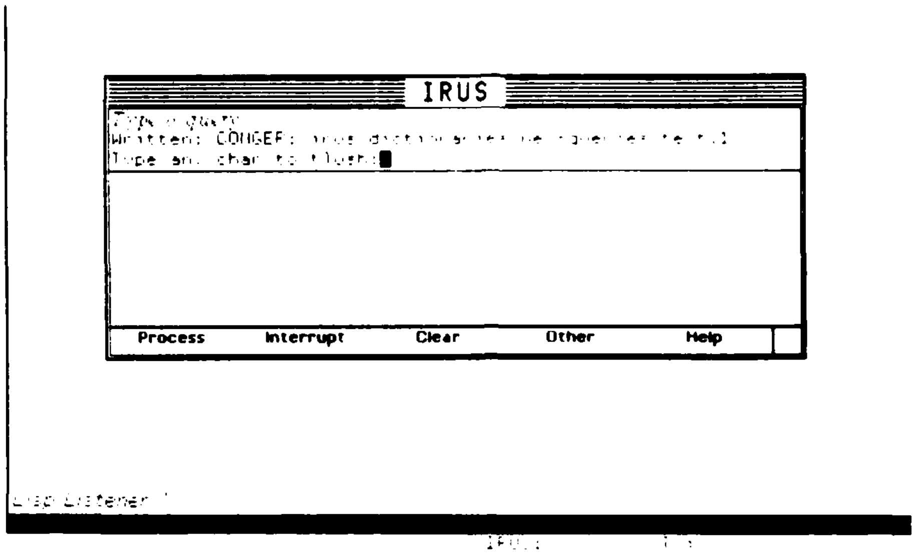
FIG. 6.   SELECT **WRITE** COMMAND



```
Type a query
list the ships in san diego

which are c4
Write queries to: (IRUS: DICTIONARIES; QUERIES.TEXT.NEWES
new-queries.text
```

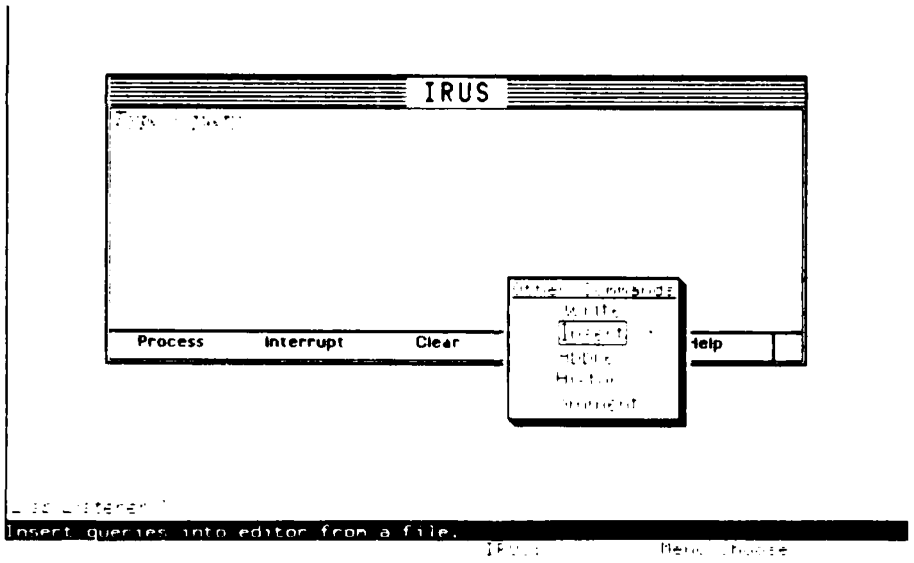
```
Lisp Listener 1
IRUS
```

FIG. 7.   PROMPT FOR QUERY FILE NAME



```
Type a query
Written: CONGER: [illegible]
Type any char to flush:
```

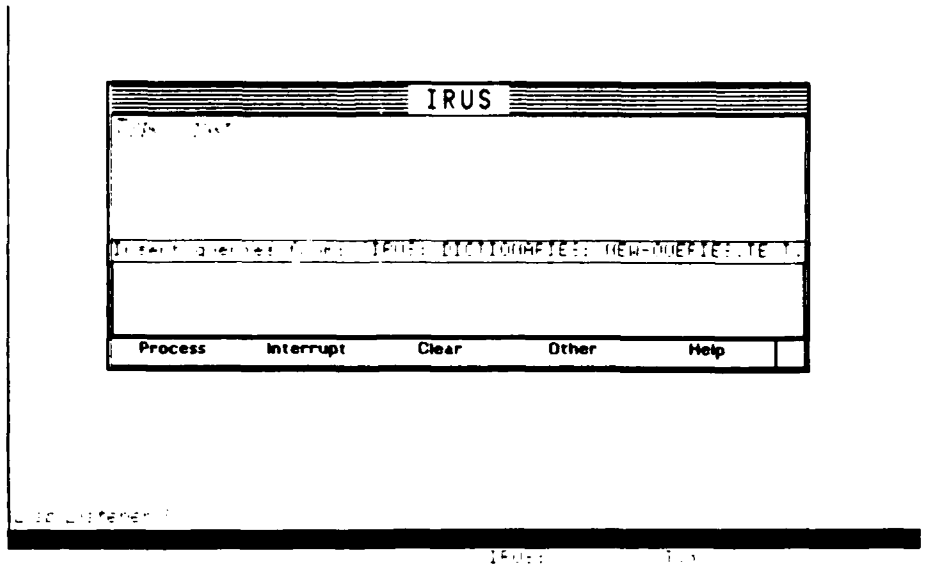
FIG. 8. CONFIRMATION OF QUERIES WRITTEN



```
Type a query
```

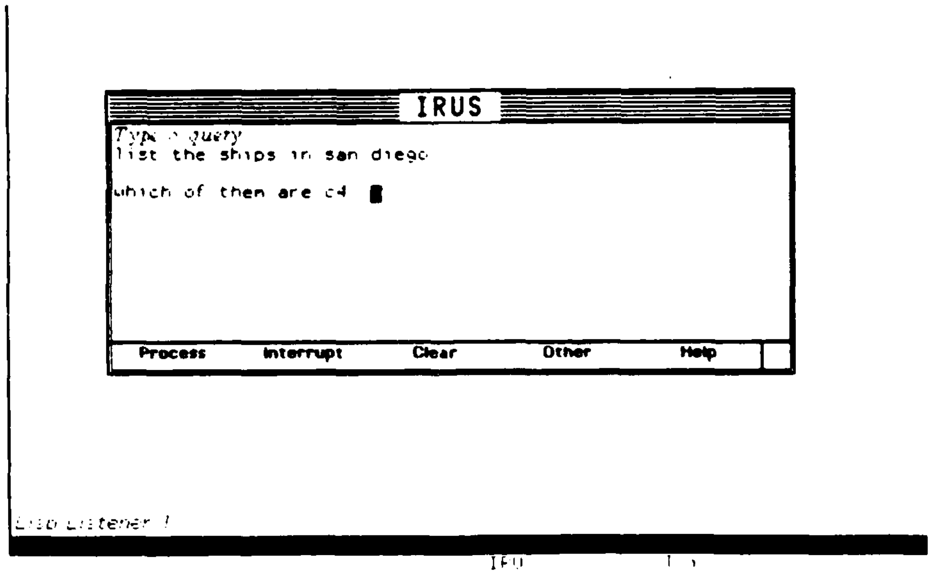
```
Insert queries into editor from a file.
```

FIG. 9. SELECT **INSERT** COMMAND



```
Insert queries from: IRUS: DICTIONARIES; NEW-QUERIES.TE[illegible]
```

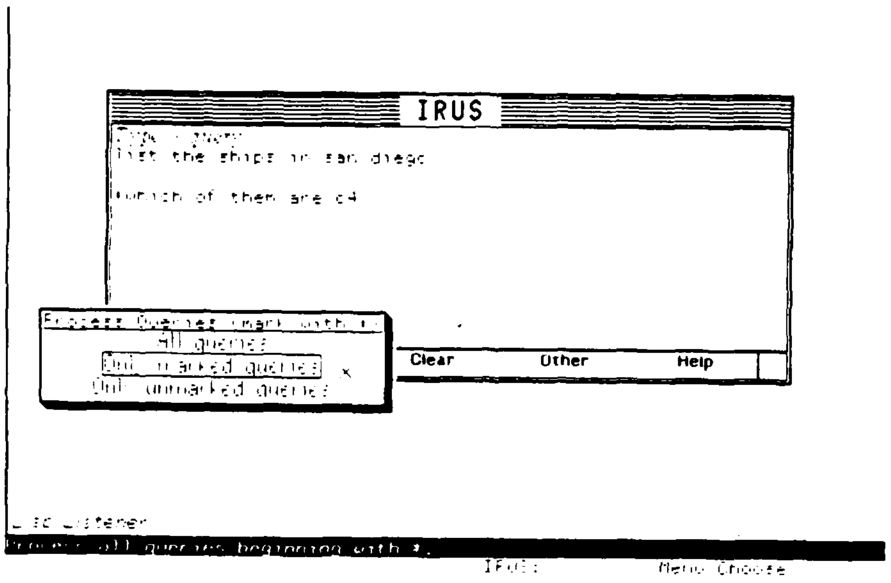
FIG. 10. PROMPT FOR QUERY FILE NAME



```
Type a query
list the ships in san diego

which of them are c4
```

FIG 11. QUERIES INSERTED FROM FILE

The end-user may want to selectively process queries from a group of queries in the Editing pane. To do this, he or she marks queries by preceeding them with an "*", clicking right on the **Process** command, and selecting one of the choices from the resulting pop-up menu: process all queries, only marked queries, or only unmarked queries (see Figure 12).

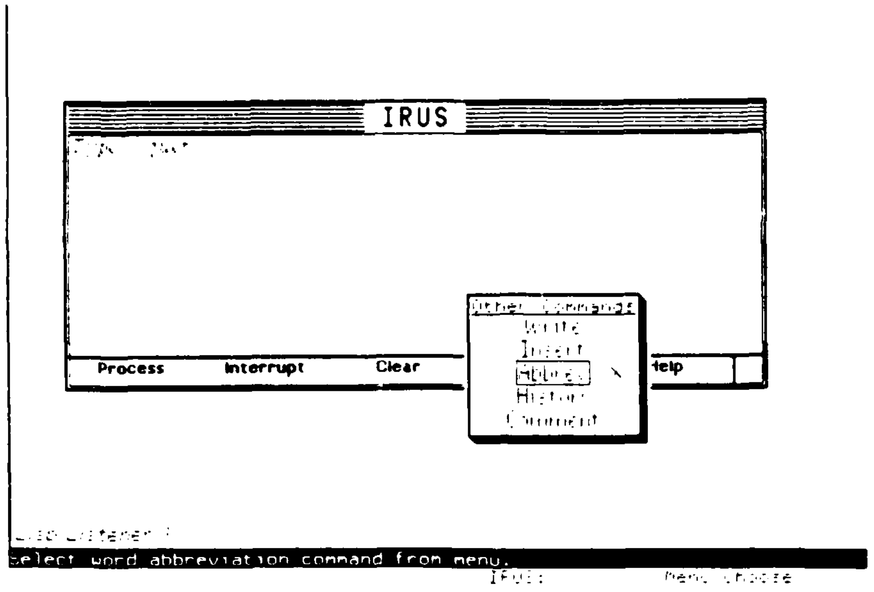


```
Type a query
list the ships in san diego

which of them are c4
```

```
Lisp Listener
Process all queries beginning with *.
IRUS:                Menu Choose
```

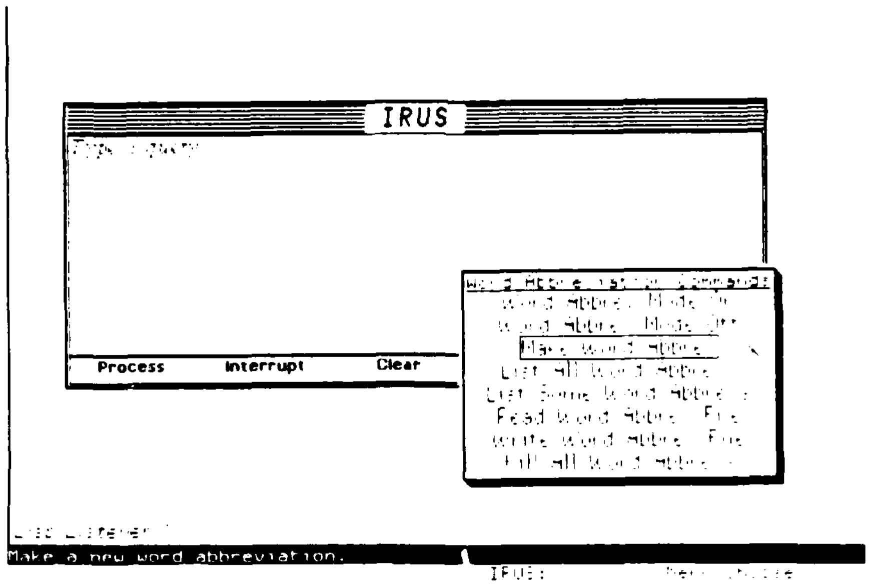
FIG 12. SELECTIVE PROCESSING OF QUERIES

The full Zmacs text editing facility of the Symbolics machine is available to the end-user for composing and editing queries. In addition to efficient text entry and editing, this editor supports macros, abbreviations, and other editing features, which can be used to provide an efficient and customizable query entry system. Thus, the port name "San Diego" (or even a whole query) might have been entered by typing only an abbreviation such as "SD", which is automatically expanded by the Zmacs editor, either by reference to a user customizable abbreviation file or to a system abbreviation file made available to all users. To create an abbreviation, the user would select the **Abbrev** command from the **Other** command menu, then select **Make Word Abbrev** from the abbreviation pop-up menu (see Figures 13 through 16).



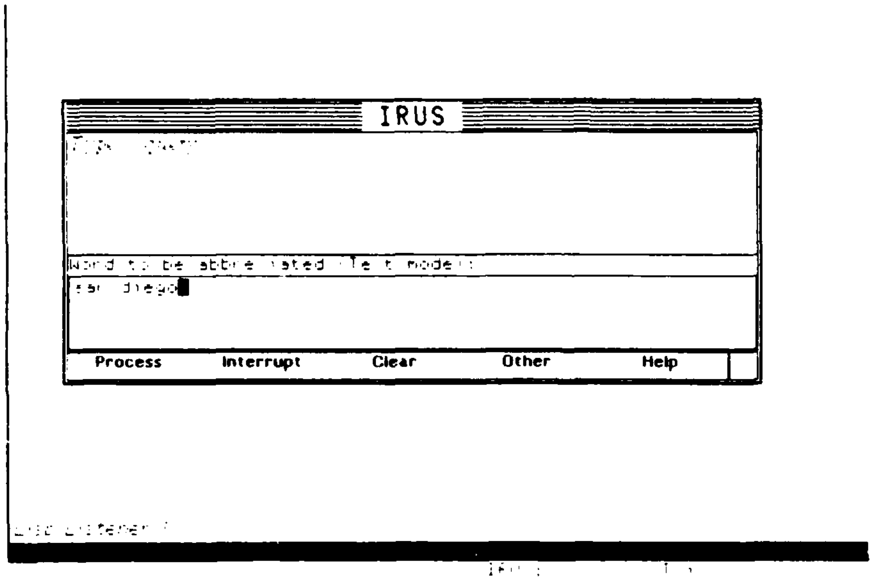
Select word abbreviation command from menu.

FIG. 13   SELECT **ABBREV** COMMAND



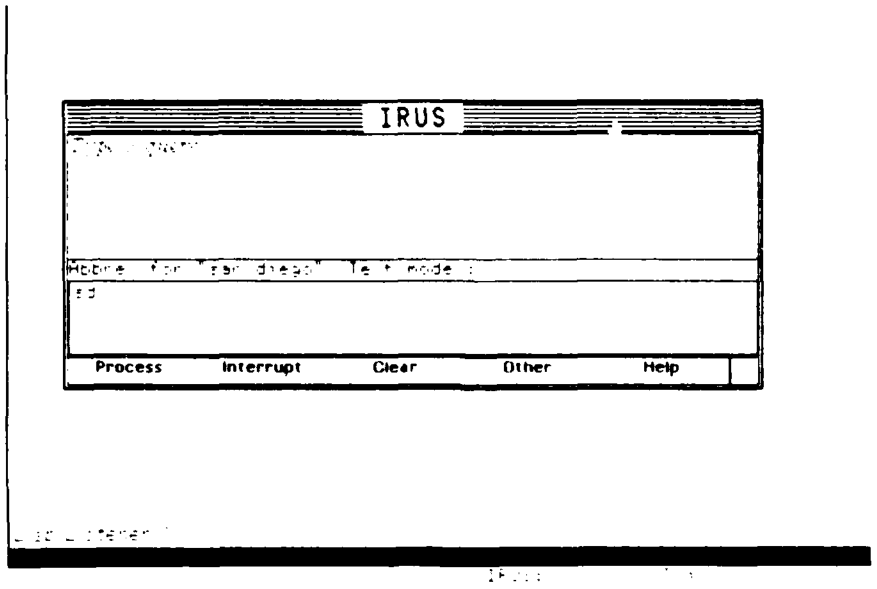
Make a new word abbreviation.

FIG. 14.   SELECT **MAKE WORD ABBREV** COMMAND



```
Word to be abbreviated (Text mode):
san diego
```

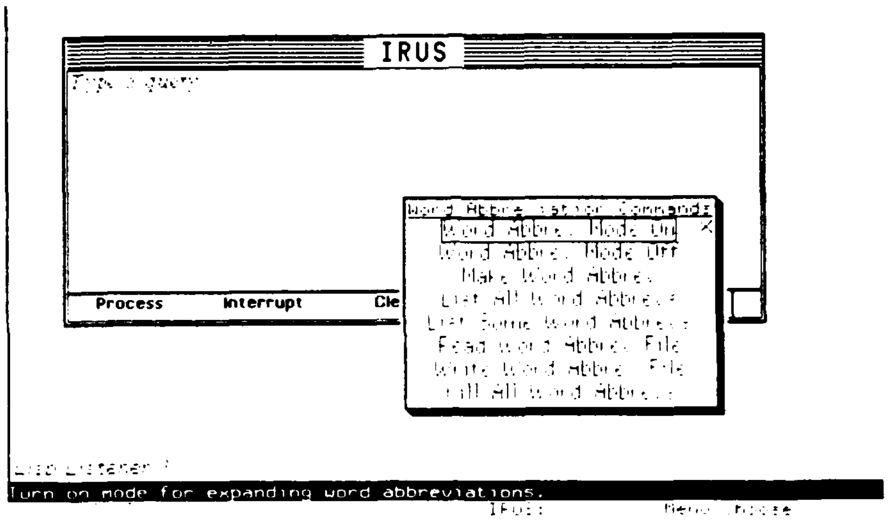
FIG. 15. PROMPT FOR WORD TO BE ABBREVIATED



```
Abbrev for "san diego" (Text mode):
sd
```

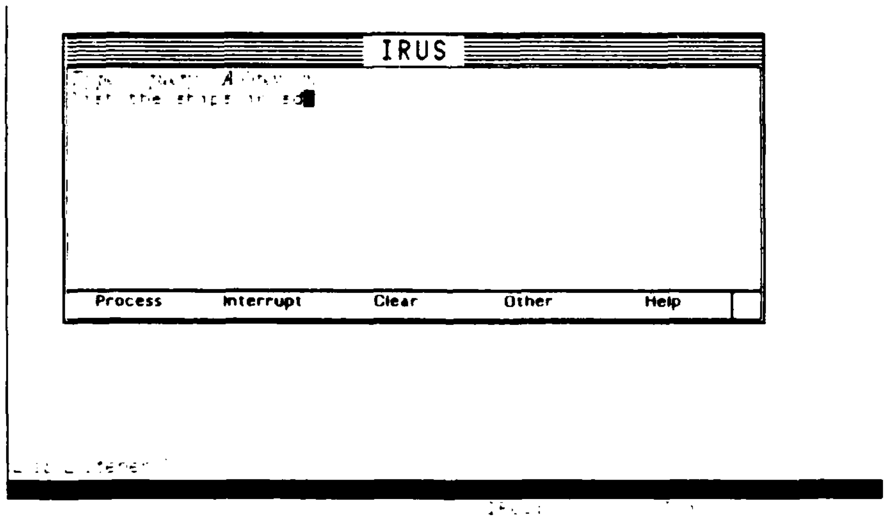
FIG. 16 PROMPT FOR ABBREVIATION

To insert the abbreviation, the word abbreviation mode is turned on by selecting **Word Abbrev Mode On** from the abbreviation pop-up menu. Then when the abbreviation is typed and followed by a space or carriage return, the abbreviation is expanded into the full word (see Figures 17 through 19).



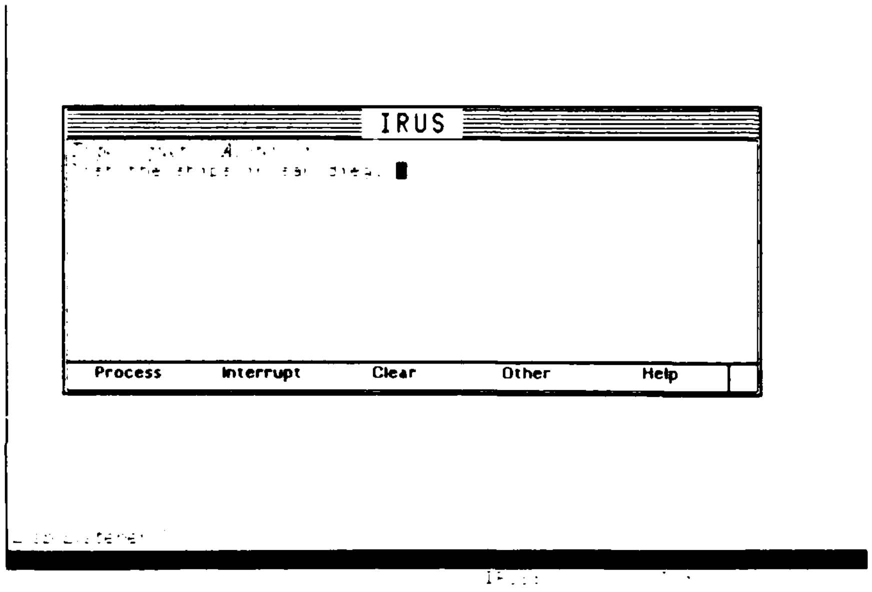
Turn on mode for expanding word abbreviations.

FIG. 17. TURN ON WORD ABBREVIATION



[illegible] the ships in sd

FIG. 18. ENTER ABBREVIATION



[illegible]
[illegible] the ships [illegible] area.

FIG. 19. WORD SUBSTITUTED FOR ABBREVIATION

Should the end-user want to see a list of queries processed thus far in the session, he or she selects the **History** command from the **Other** command menu, and a pop-up history window appears (see Figure 20). Queries then can be entered into the Editing pane not only by using the keyboard, but by selecting one of the queries in the pop-up History window (see Figures 21 through 23). Queries also could be entered using any combination of typing, reading a file, or selecting a graphic object with the mouse. The query, "List the ships in San Diego", could have been composed by reading a help file or menu of possible queries and selecting "list", typing "the ships", and selecting the visual object corresponding to the port "San Diego" on a chart display, provided that the graphic objects or pieces of text are mouse selectable and return a string.

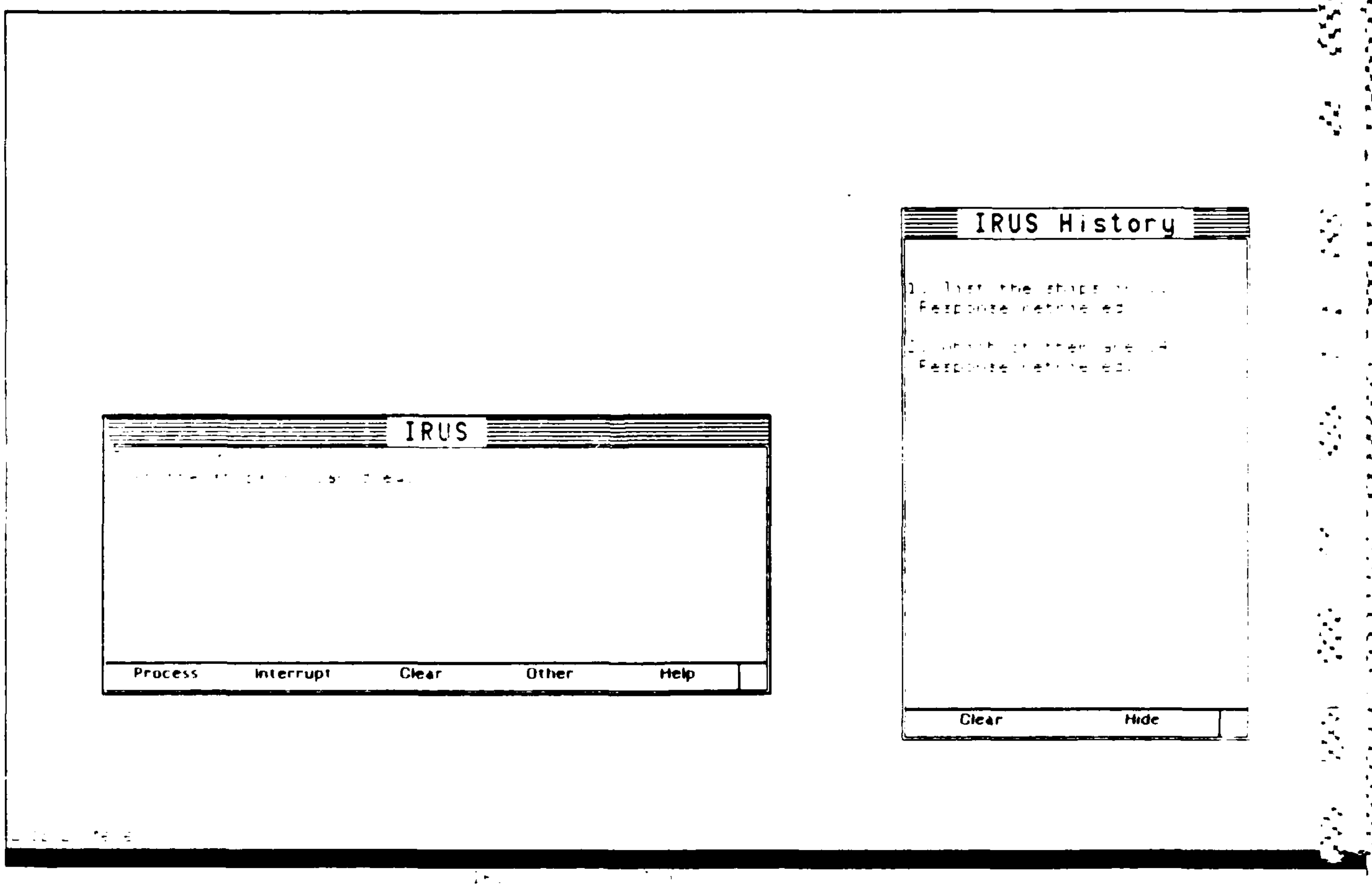
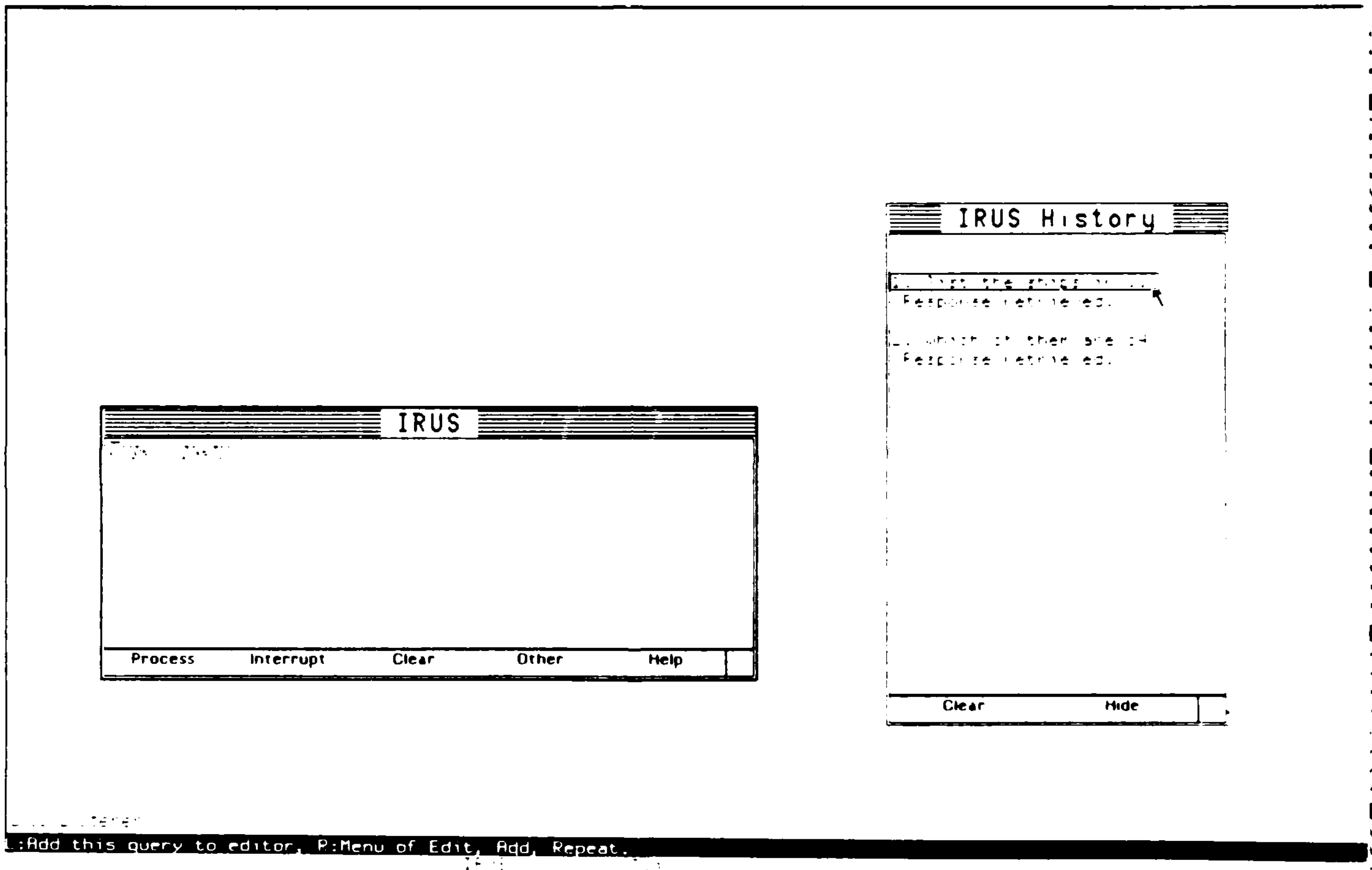


FIG 20 POP-UP HISTORY WINDOW

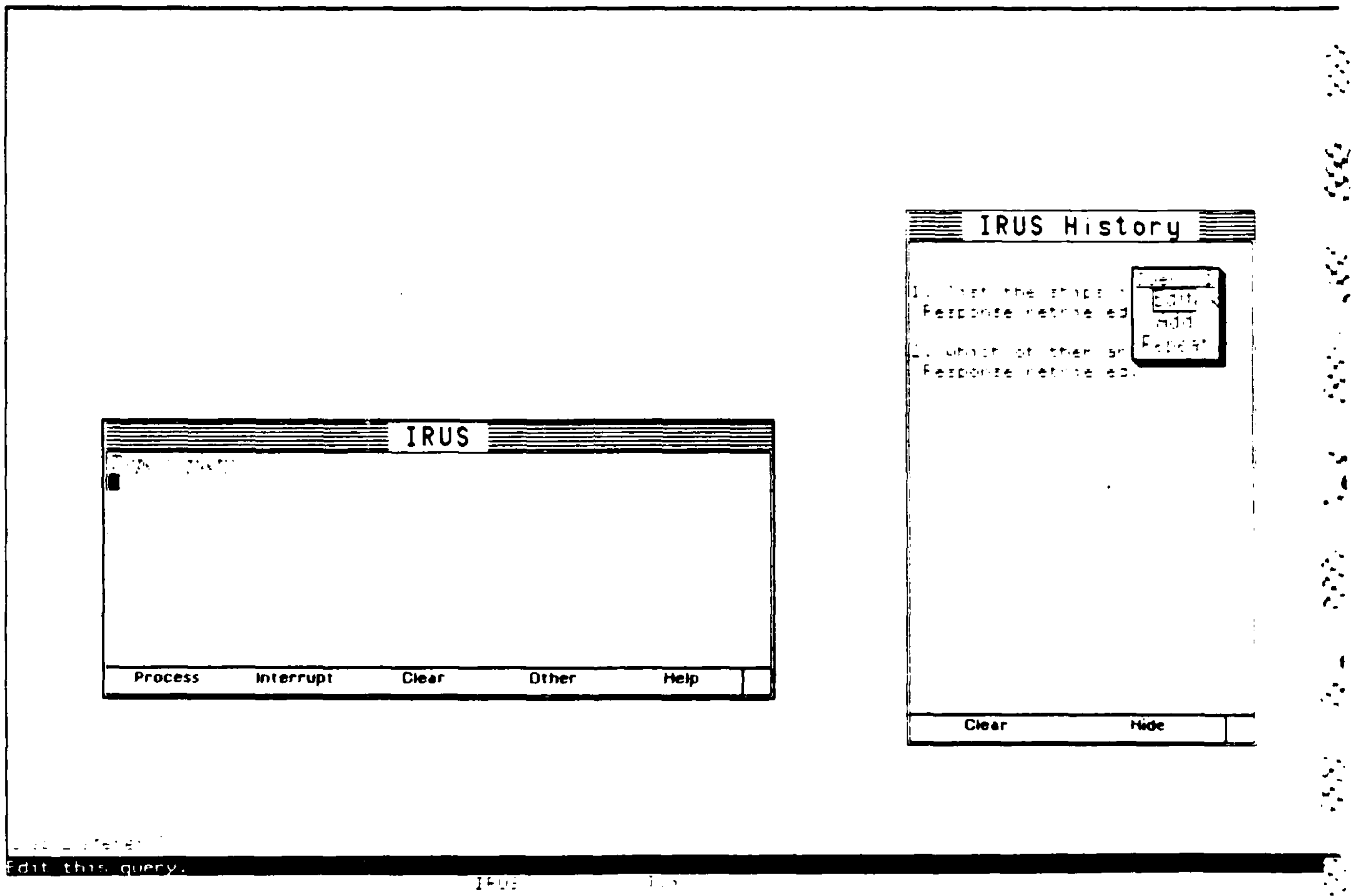


[illegible]
Response retrieved.

[illegible] which of them are [illegible]
Response retrieved.

L:Add this query to editor, R:Menu of Edit, Add, Repeat.

FIG. 21 SELECT QUERY FROM HISTORY

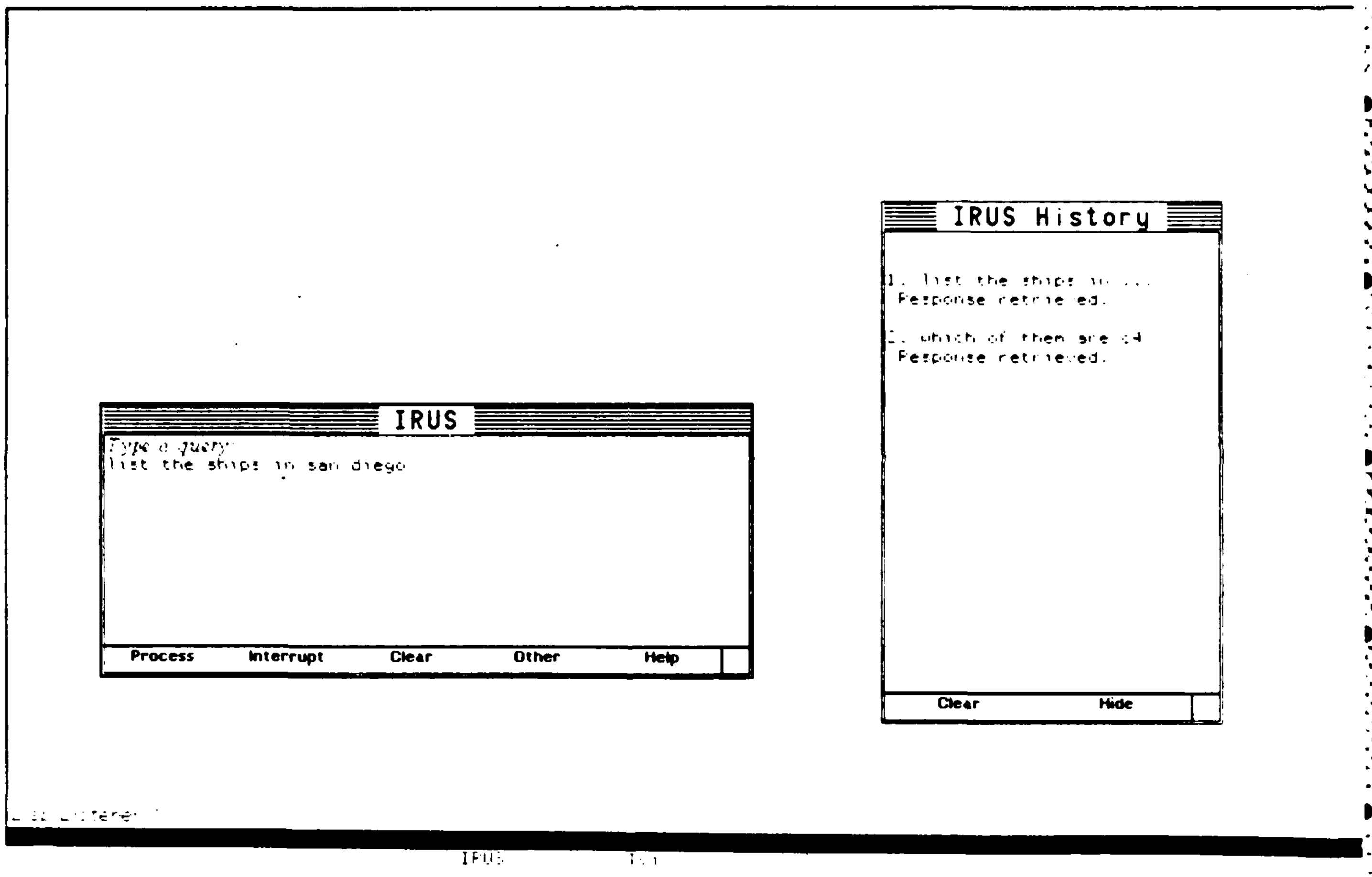


1. [illegible]
Response retrieved.

2. [illegible]
Response retrieved.

Edit this query.

FIG. 22. SELECT **EDIT** COMMAND FOR QUERY



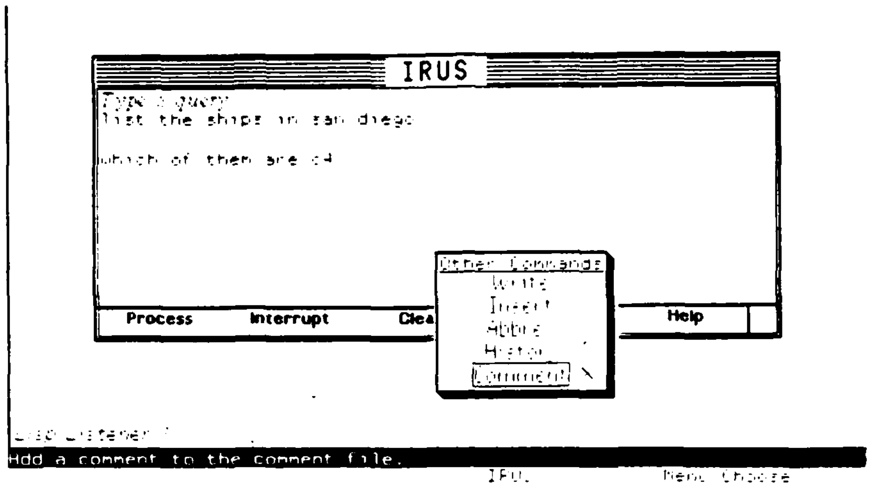
Type a query:
list the ships in san diego

1. list the ships in ...
 Response retrieved.

2. which of them are c4
 Response retrieved.

FIG. 23. QUERY ADDED TO EDITOR PANE

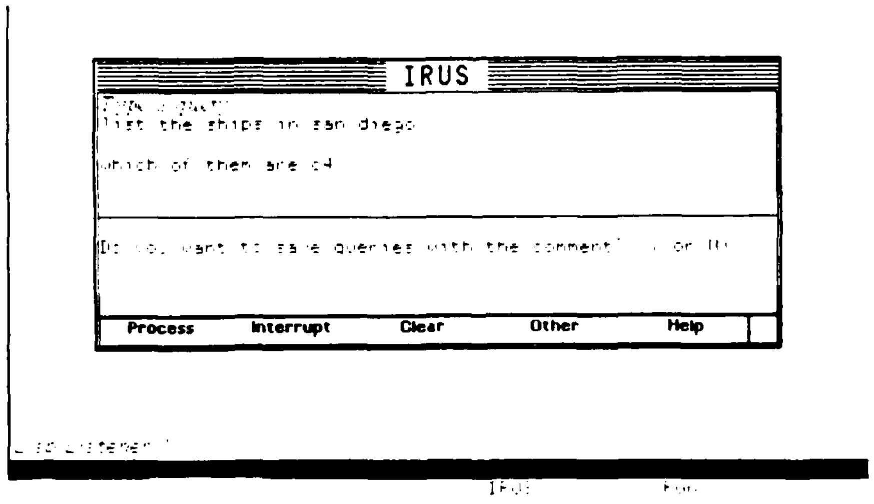
Finally, a facility for saving comments is available by selecting **Comment** from the **Other** command menu (see Figures 24 through 27). This feature can be used, for example, to save comments about a certain query along with the query or to provide feedback to the developers about using the system. Alternatively, a separate facility could be provided that, when invoked, would automatically send a message to the developer or system maintainer. (The Lisp machine mail system provides one such facility, messages are sent to Bug-Lispm.)



```
list the ships in san diego

which of them are c4
```

```
Add a comment to the comment file.
```

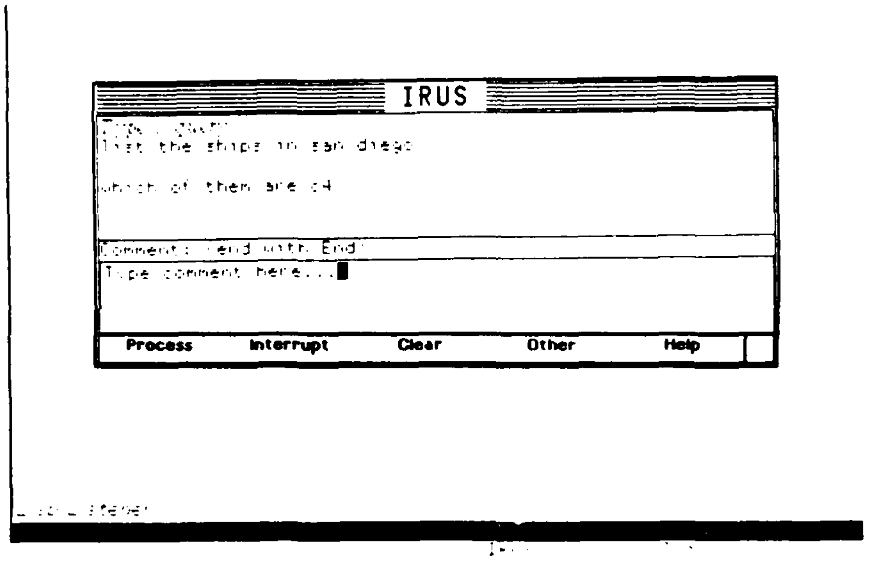
FIG. 24. SELECT **COMMENT** COMMAND



```
list the ships in san diego

which of them are c4
```

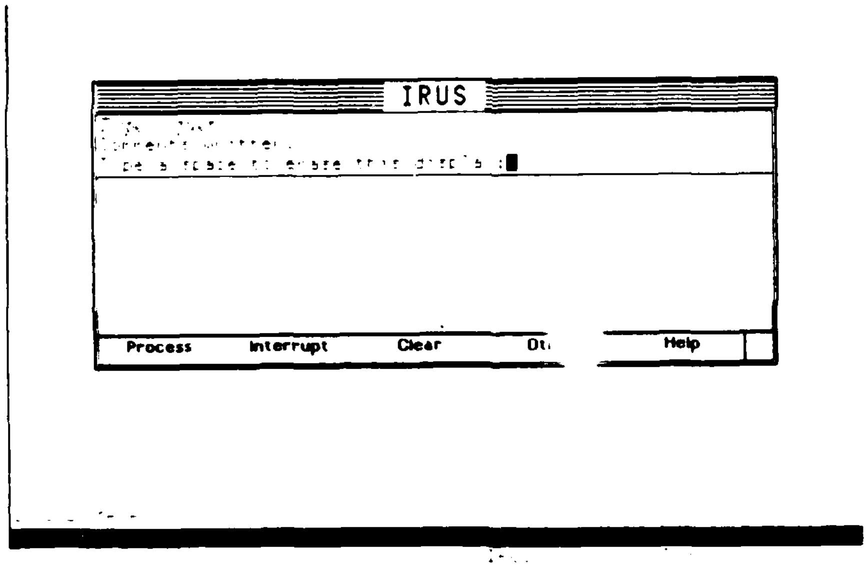
```
Do you want to save queries with the comment? (Y or N)
```

FIG. 25 PROMPT FOR SAVING QUERIES IN COMMENT



```
[illegible]
list the ships in san diego

which of them are c4

Comment: (end with End)
Type comment here...
```

FIG. 26. ENTER COMMENT (QUERIES NOT SAVED)



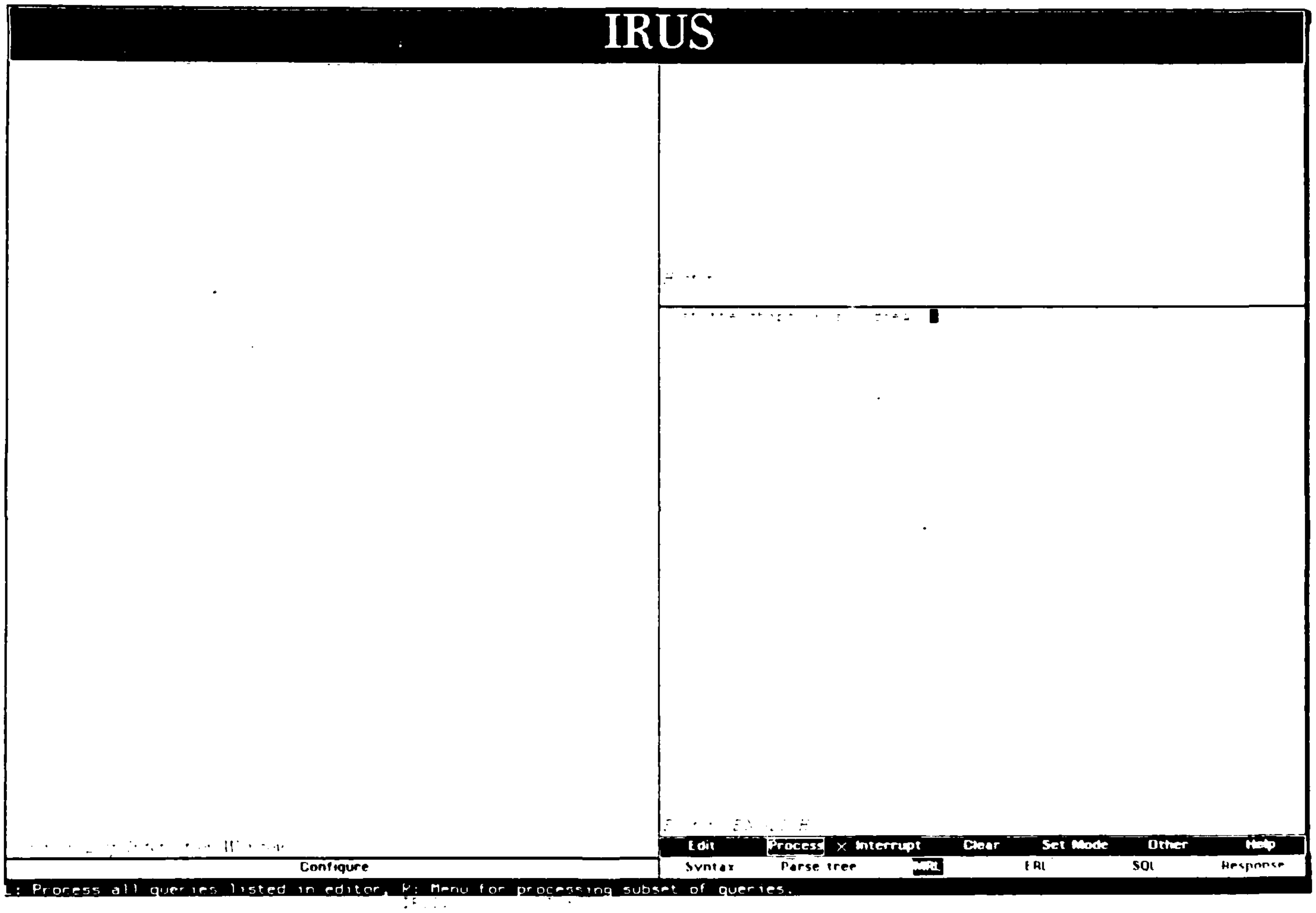
```
[illegible]
Comments written.
Type a space to erase this display:
```

FIG 27. CONFIRMATION OF COMMENTS WRITTEN

5.2 Developer Interaction

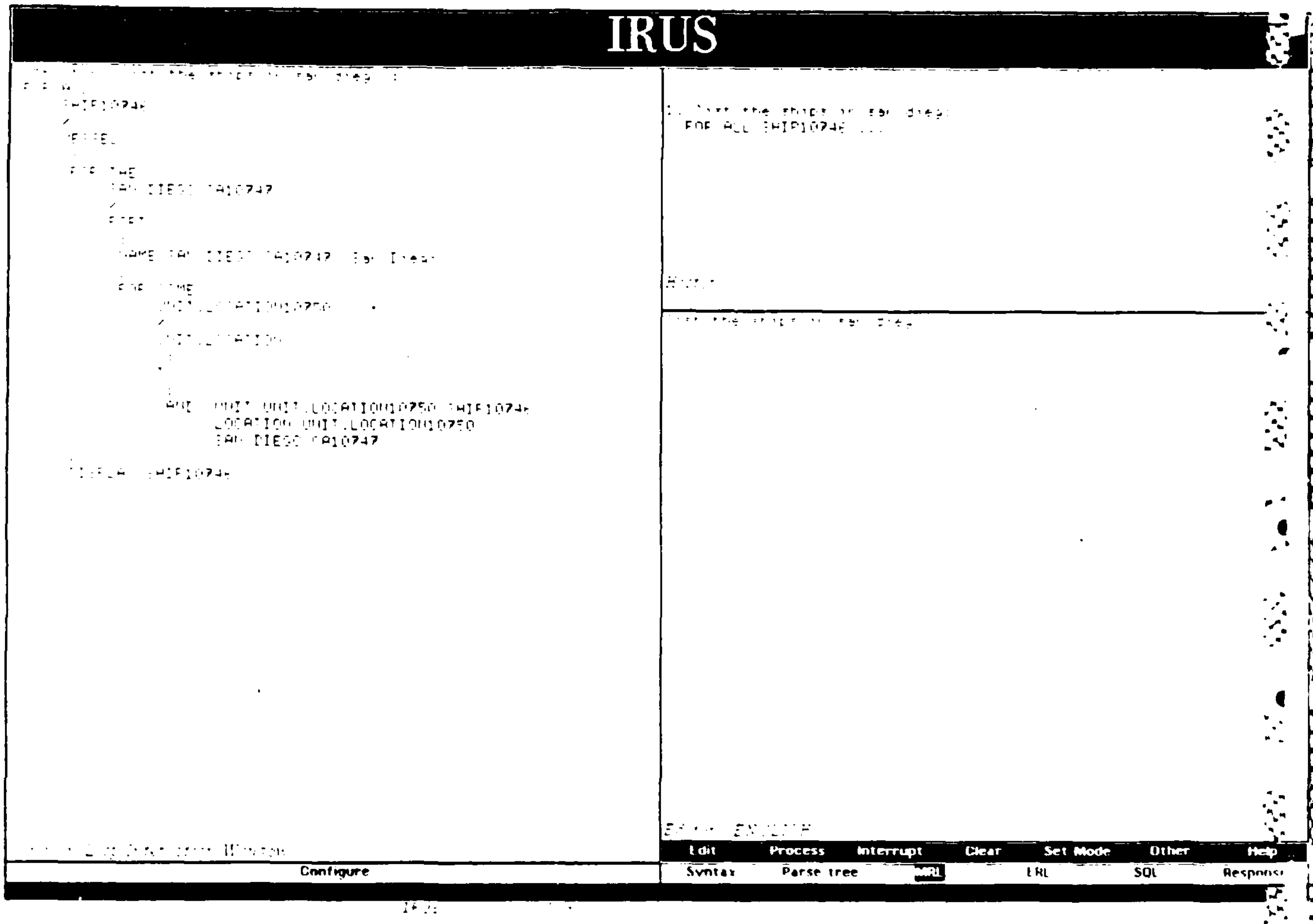
All the functionality supplied to the end-user is available to the developer. In addition, capabilities have been provided for system development, debugging, testing, and updating IRUS. The following scenario illustrates the use of these capabilities.

The full screen developer configuration is selected, and a query is entered into the Editing pane (see Figure 28). The developer is often interested in stopping the processing at an intermediate stage and may select a stage on the processing stages menu located just below the Editing pane. In this example, the MRL stage has been selected. The query is processed, and the results are shown in the Interaction pane and in an abbreviated form in the History pane (see Figure 29). The result, MRL in this example, can be edited by clicking on the word "MRL" in the Interaction pane, then clicking on **Edit** in the menu that pops up. The MRL Lisp expression is added automatically to the Editing pane, and the mode is set to MRL. (See Figures 30 through 32.) Should the developer then want to proceed with processing to the next step, ERL can be selected on the processing stages menu, and the query processed. The resulting ERL is displayed in the Interaction pane and in abbreviated form in the History pane (see Figure 33).



L: Process all queries listed in editor, R: Menu for processing subset of queries.

FIG 28 PROCESS QUERY IN DEVELOPER CONFIGURATION

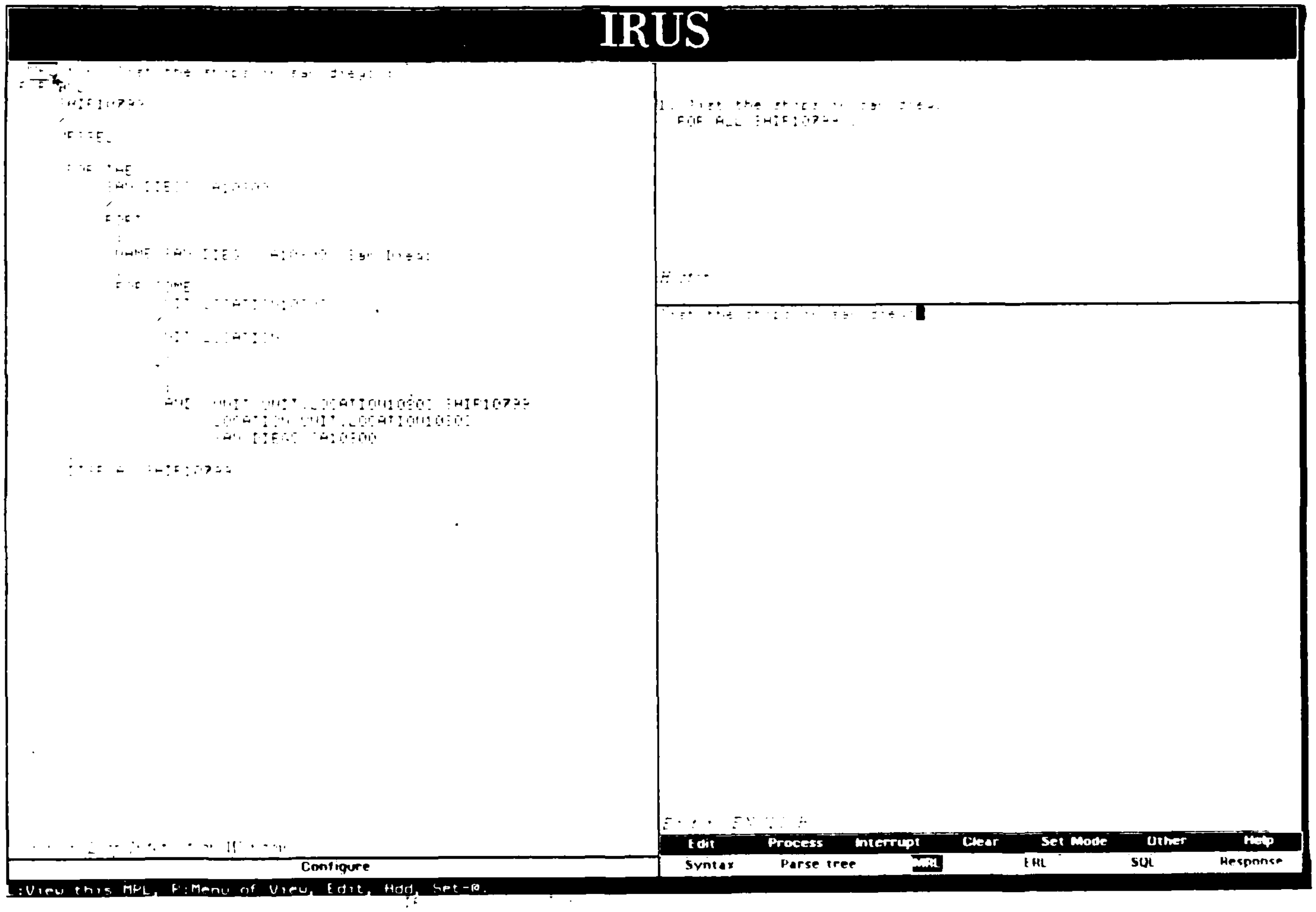


```
[illegible] the ships in San Diego
FOR ALL
    SHIP10746
    /
    [illegible]
    FOR THE
        SAN DIEGO CA10747
        /
        [illegible]
          NAME SAN DIEGO CA10747 San Diego
          FOR THE
              UNIT.LOCATION10750
              /
              UNIT.LOCATION
              [illegible]
              AND (UNIT UNIT.LOCATION10750 SHIP10746
                   LOCATION UNIT.LOCATION10750
                   SAN DIEGO CA10747
    DISPLAY SHIP10746
```

```
1. List the ships in San Diego
  FOR ALL SHIP10746 ...
```

```
List the ships in San Diego
```

FIG. 29. QUERY RESULT DISPLAYED IN INTERACTION AND HISTORY PANES



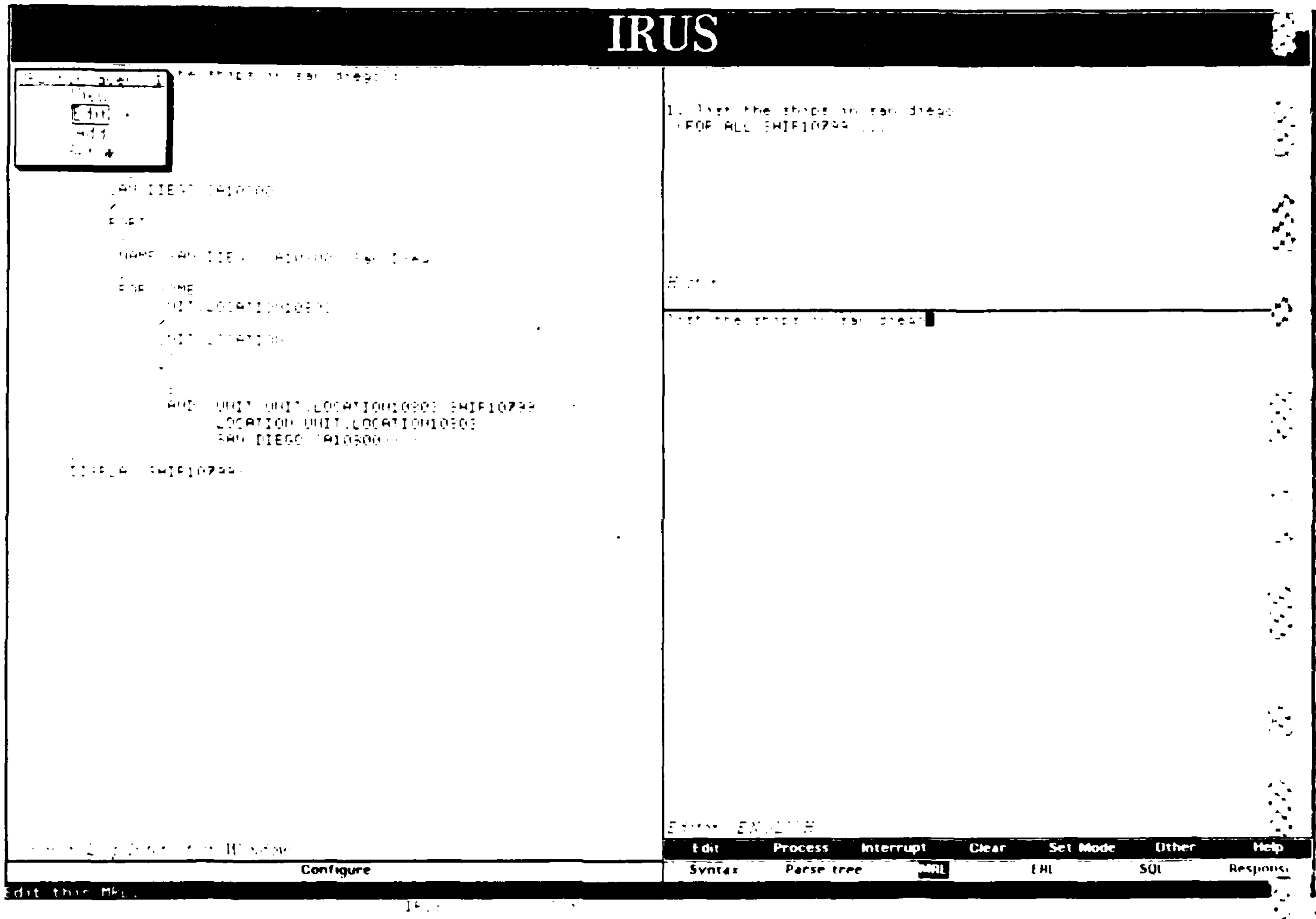
```
[illegible]
  SHIP10799
  /
  [illegible]

  [illegible] THE
      [illegible]
      /
      [illegible]

       NAME [illegible] San Diego

       [illegible] SOME
           [illegible]
          /
           [illegible]

           AND  [illegible] UNIT.LOCATION10301 SHIP10799
                LOCATION UNIT.LOCATION10301
                [illegible]

  [illegible] SHIP10799
```

```
1. [illegible] San Diego.
   FOR ALL SHIP10799 [illegible]
```

```
[illegible] San Diego.
```

```
:View this MRL, R:Menu of View, Edit, Add, Set-@.
```

FIG 30 SELECT MRL FROM INTERACTION PANE

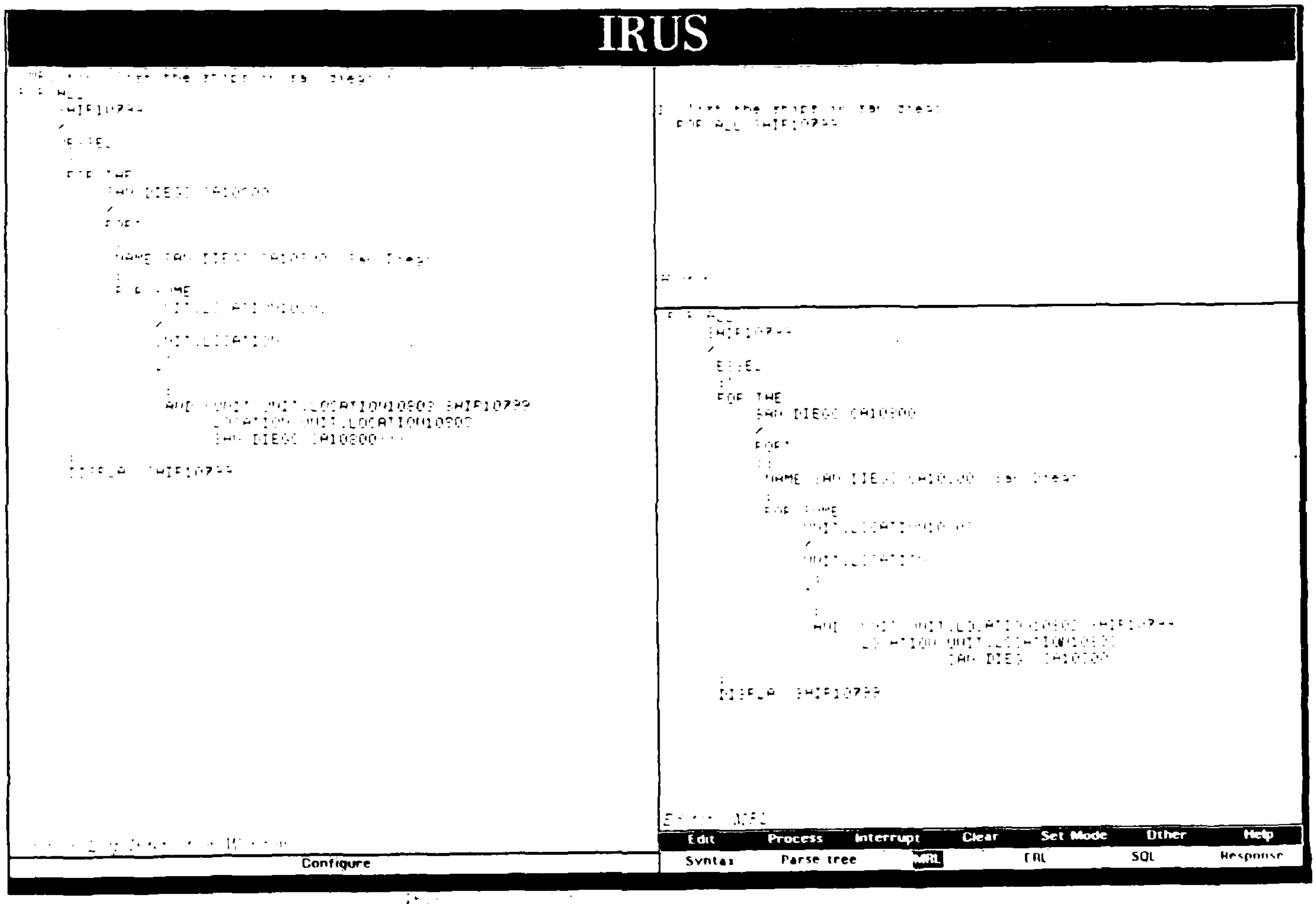


```
[illegible] the ships in san diego

SAN DIEGO [illegible]
/
[illegible]

NAME SAN DIE[illegible]  [illegible]

[illegible]
     UNIT.LOCATION[illegible]
   /
     UNIT.LOCATION[illegible]

     AND  UNIT UNIT.LOCATION10903 SHIP10799
          LOCATION UNIT.LOCATION10903
          SAN DIEGO [illegible]A10800

[illegible] SHIP10799
```

```
1. list the ships in san diego
 (FOR ALL SHIP10799 ...

list the ships in san diego
```

Edit this MRL.

FIG 31   SELECT EDIT COMMAND FOR MRL

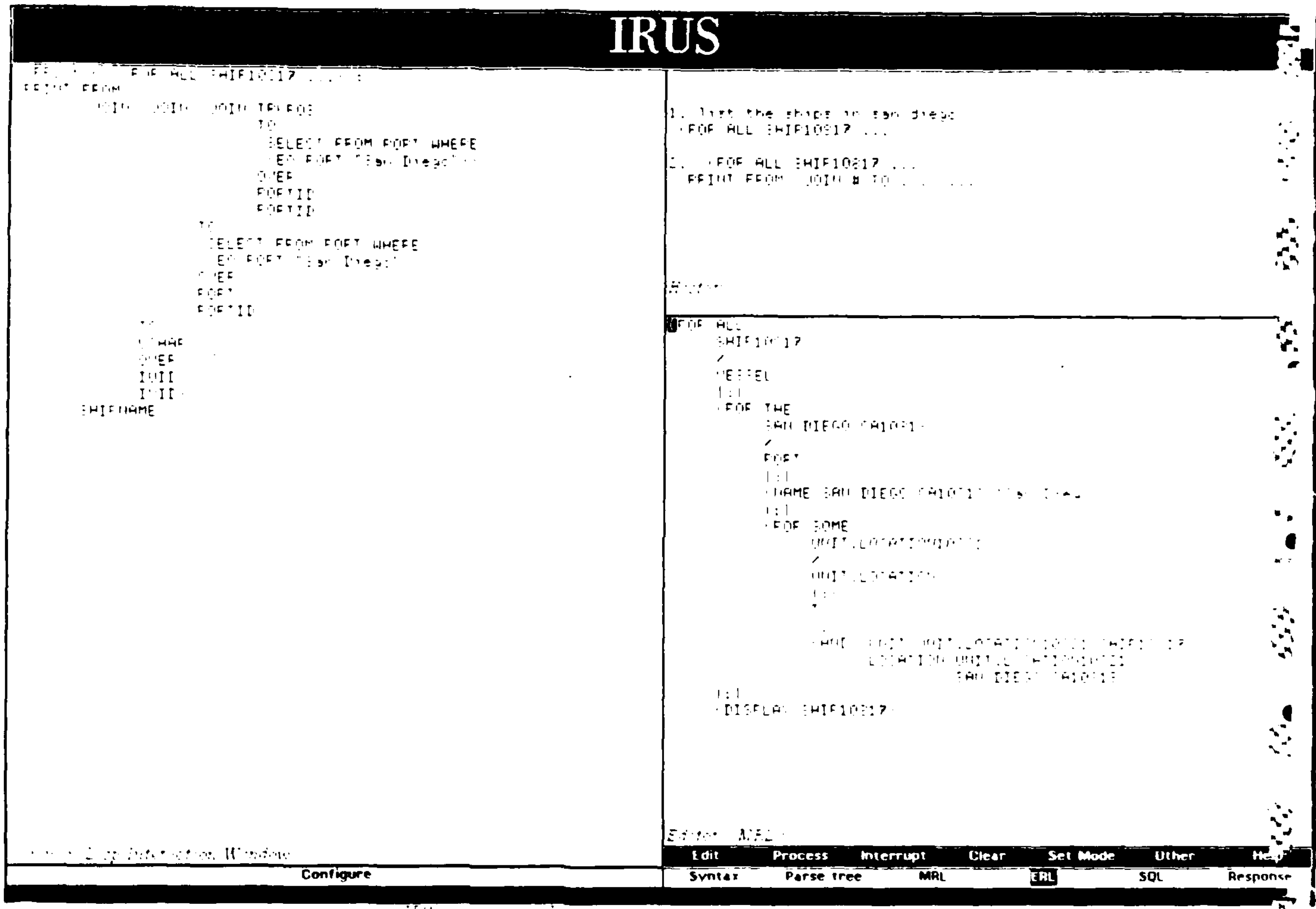


```
[illegible] List the ships in San Diego [illegible]
FOR ALL
    SHIP10799
    /
    VESSEL
    :
    FOR THE
        SAN DIEGO CA10800
        /
        PORT
        :
         NAME SAN DIEGO CA10800 San Diego
         :
         FOR SOME
             [illegible]
             /
             UNIT.LOCATION
             :
             .
              :
              AND [illegible] UNIT.LOCATION10802 SHIP10799
                    LOCATION UNIT.LOCATION10802
                    SAN DIEGO CA10800 [illegible]
    :
    DISPLAY SHIP10799
```

```
: List the ships in San Diego
  FOR ALL SHIP10799
```

```
FOR ALL
    SHIP10799
    /
    VESSEL
    :
    FOR THE
        SAN DIEGO CA10800
        /
        PORT
        :
         NAME SAN DIEGO CA10800 San Diego
         :
         FOR SOME
             UNIT.LOCATION10802
             /
             UNIT.LOCATION
             :
             .
              :
              AND [illegible] UNIT.LOCATION10802 SHIP10799
                    LOCATION UNIT.LOCATION10802
                                SAN DIEGO CA10800
    :
    DISPLAY SHIP10799
```

FIG. 32. MRL ADDED TO EDITOR PANE



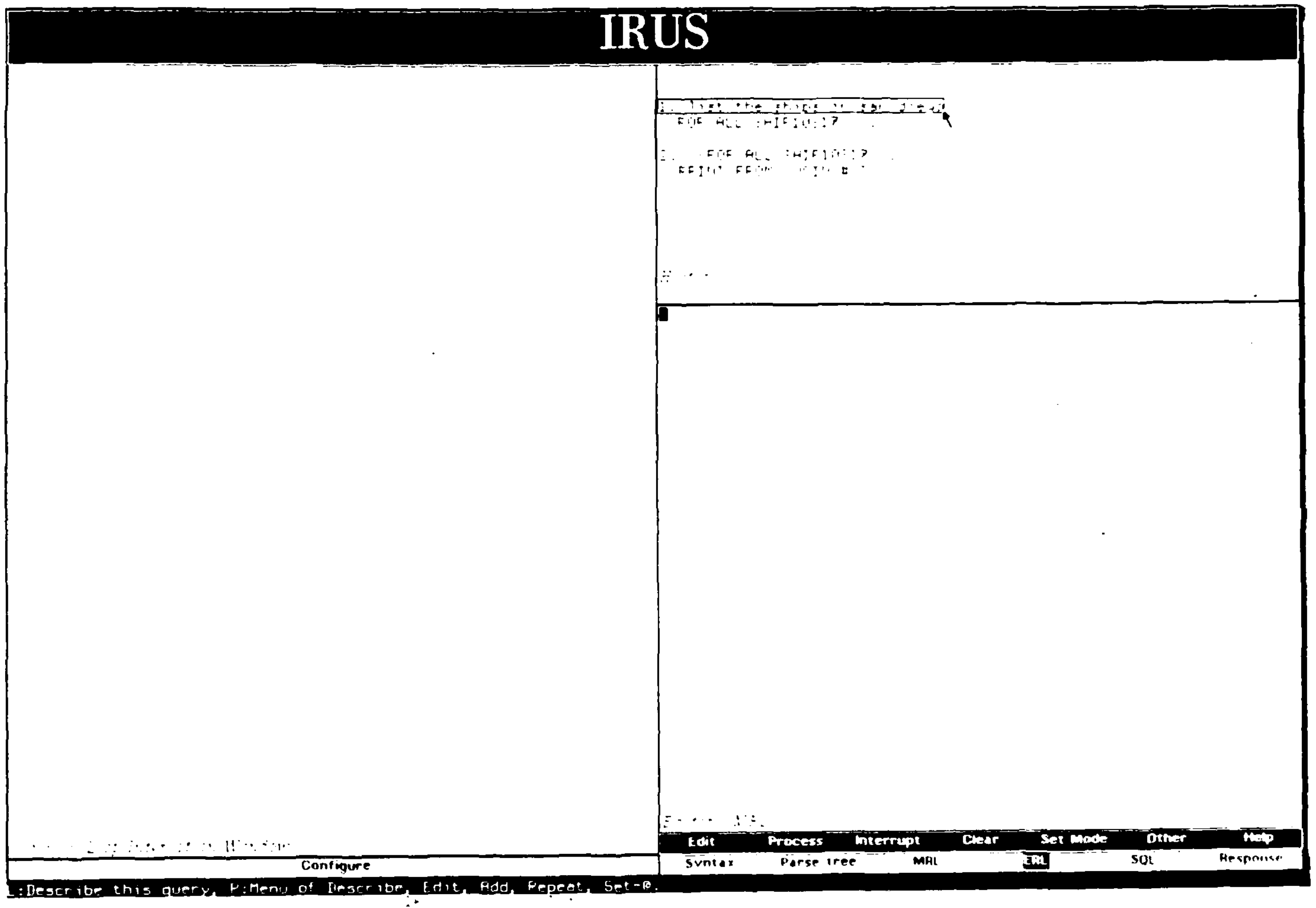
```
[illegible] (FOR ALL SHIP10217 ...) :
PRINT FROM
        JOIN  JOIN  JOIN TRACKS
                         TO
                          SELECT FROM PORT WHERE
                           EQ PORT "San Diego"
                         OVER
                         PORTID
                         PORTID
                  TO
                   SELECT FROM PORT WHERE
                    EQ PORT "San Diego"
                  OVER
                  PORT
                  PORTID
             TO
             UCHAR
             OVER
             UUII
             UUII
       SHIPNAME
```

```
1. list the ships in san diego
 (FOR ALL SHIP10217 ...

2.  (FOR ALL SHIP10217 ...
  PRINT FROM  JOIN # TO . . . ...
```

```
(FOR ALL
     SHIP10217
     /
     VESSEL
     |:|
     (FOR THE
          SAN DIEGO CA10071)
          /
          PORT
          |:|
          (NAME SAN DIEGO CA10071 "San Diego")
          |:|
          (FOR SOME
               UNIT.LOCATION[illegible]
               /
               UNIT.LOCATION
               |:|

               (AND [illegible]
                     LOCATION UNIT.LOCATION[illegible]
                              SAN DIEGO CA10071
     |:|
     (DISPLAY SHIP10217)
```

FIG. 33 ERL DISPLAYED IN INTERACTION AND HISTORY PANES

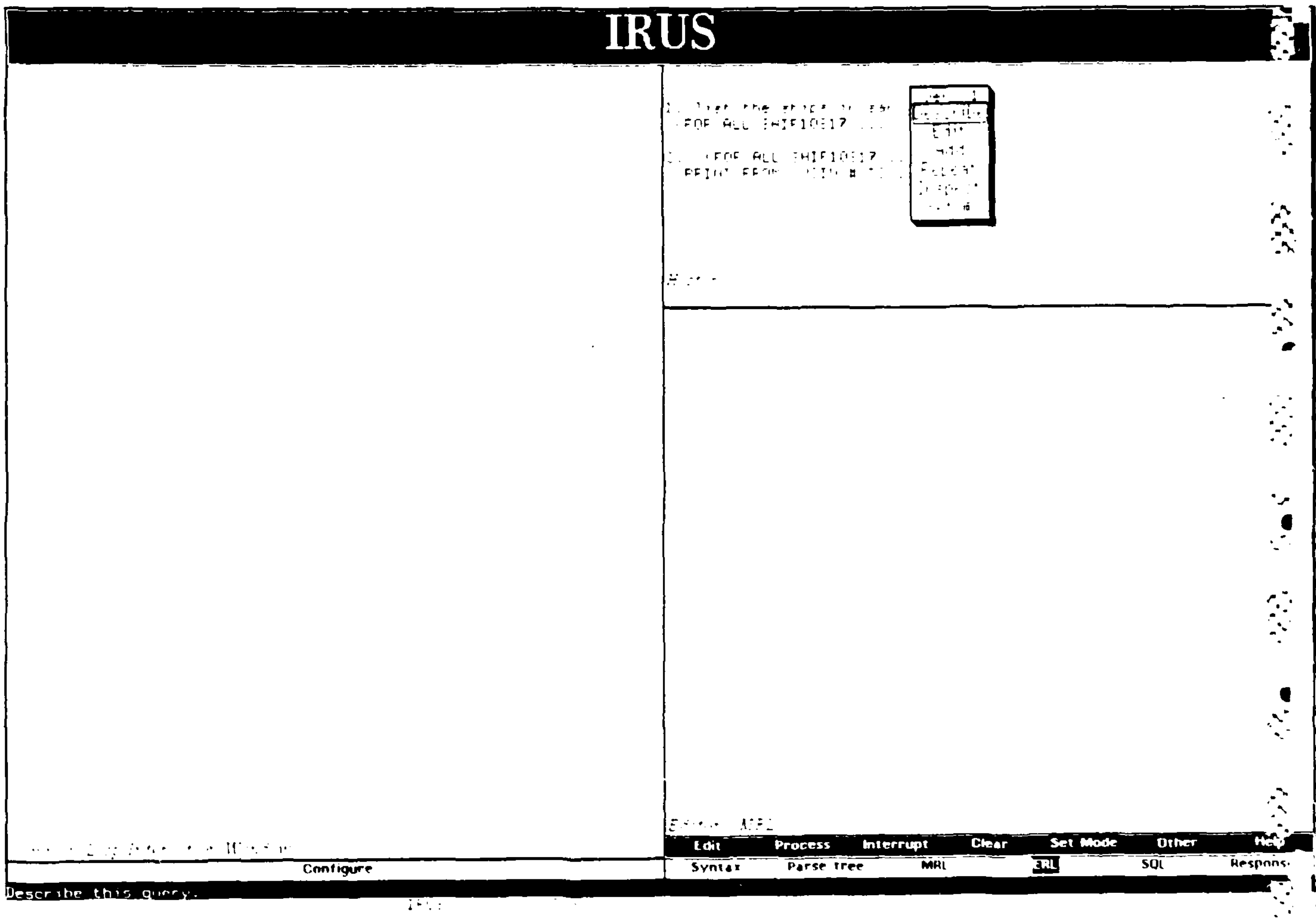
The developer can display all processed stages of a query by clicking on the query in the History pane (see Figures 34 through 36). The query stages, displayed in the Interaction pane, are mouse-sensitive and can be operated upon (e.g. edited or viewed in more detail) by selecting an operation from a pop-up menu that appears when the stage is selected with the mouse (see Figures 37 through 39).



```
1. [illegible]
   FOR ALL SHIP[illegible]?

2. FOR ALL SHIP[illegible]?
   PRINT FROM [illegible]
```

L:Describe this query, R:Menu of Describe, Edit, Add, Repeat, Set-@.

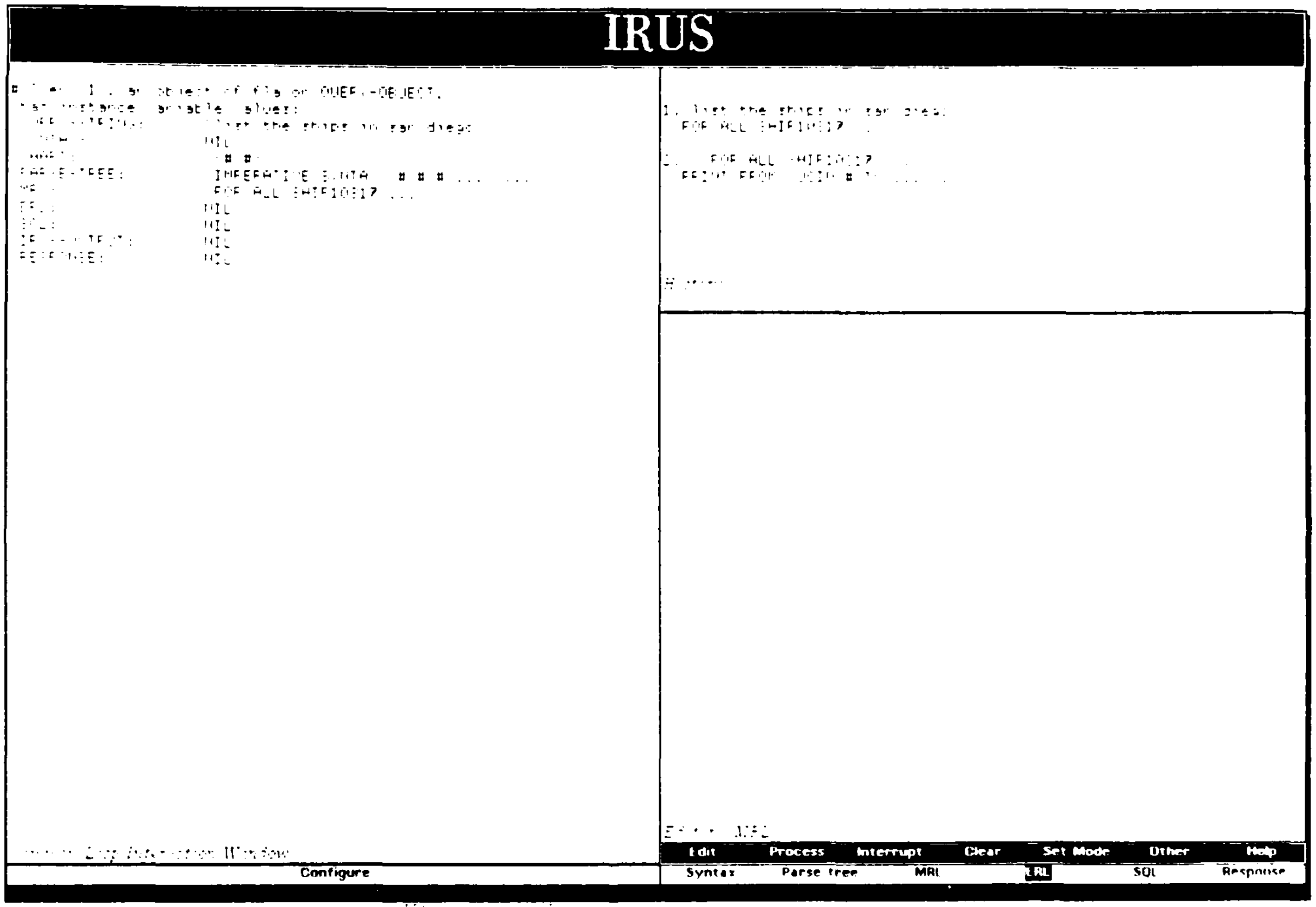
FIG. 34. S.LECT QUERY FROM HISTORY PANE



```
1. [illegible]
   (FOR ALL [illegible] ...
2. (FOR ALL [illegible] ...
   PRINT [illegible]
```

Describe this query.

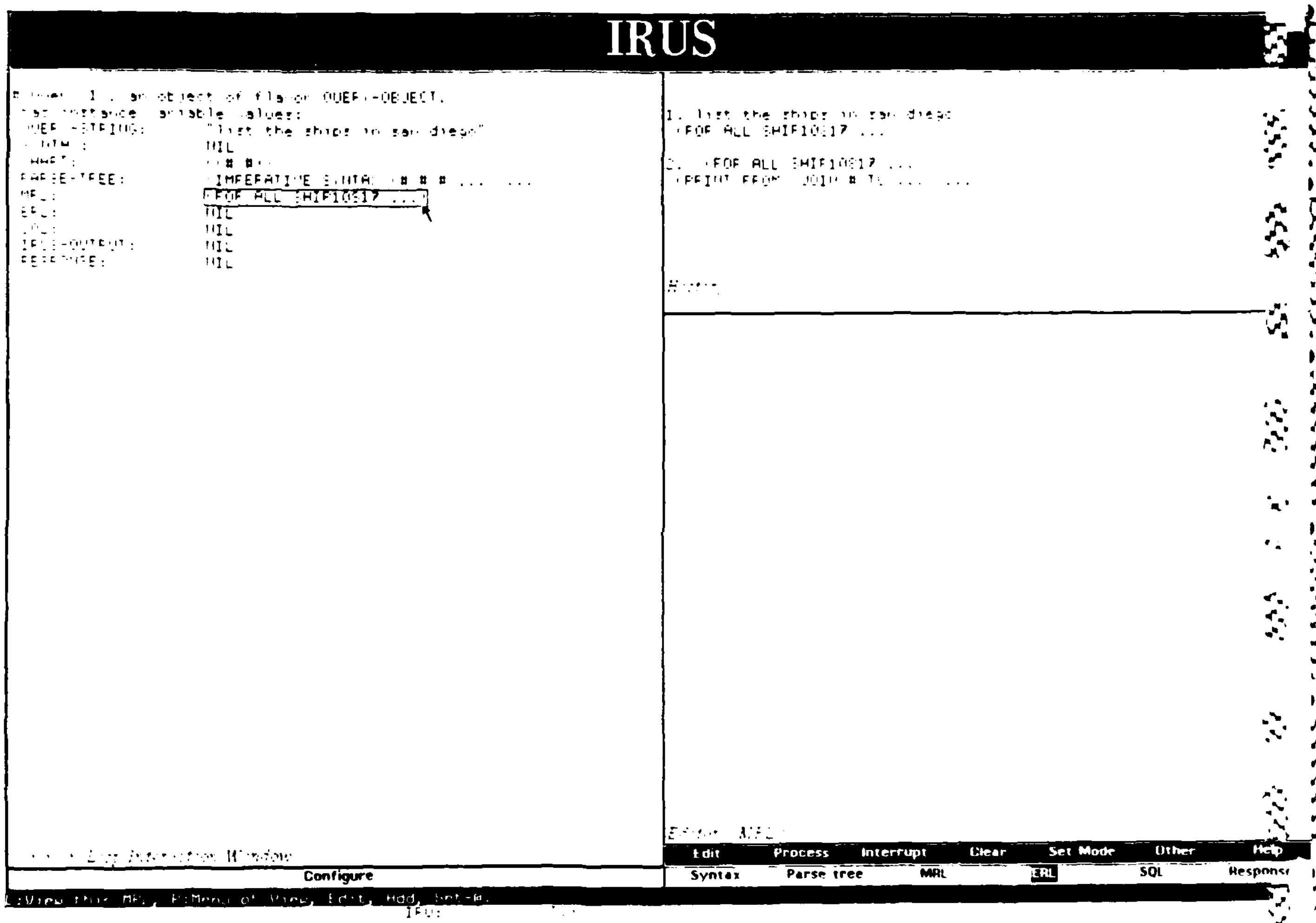
FIG. 35 SELECT DESCRIBE COMMAND FOR QUERY



```
#[illegible] 1, an object of flavor QUERY-OBJECT,
has instance variable values:
  [illegible]:            "list the ships in san diego
  [illegible]:            NIL
  [illegible]:             # #
  PARSE-TREE:              IMPERATIVE SYNTA[illegible]  # # # ...  ...
  MRL:                     FOR ALL SHIP10317 ...
  ERL:                    NIL
  SQL:                    NIL
  [illegible]:            NIL
  RESPONSE:               NIL
```

```
1. list the ships in san diego
  FOR ALL SHIP10317 ...

[illegible]   FOR ALL SHIP10317
  PRINT FROM [illegible] # [illegible] ...
```

FIG. 36. QUERY OBJECT DESCRIBED IN INTERACTION PANE

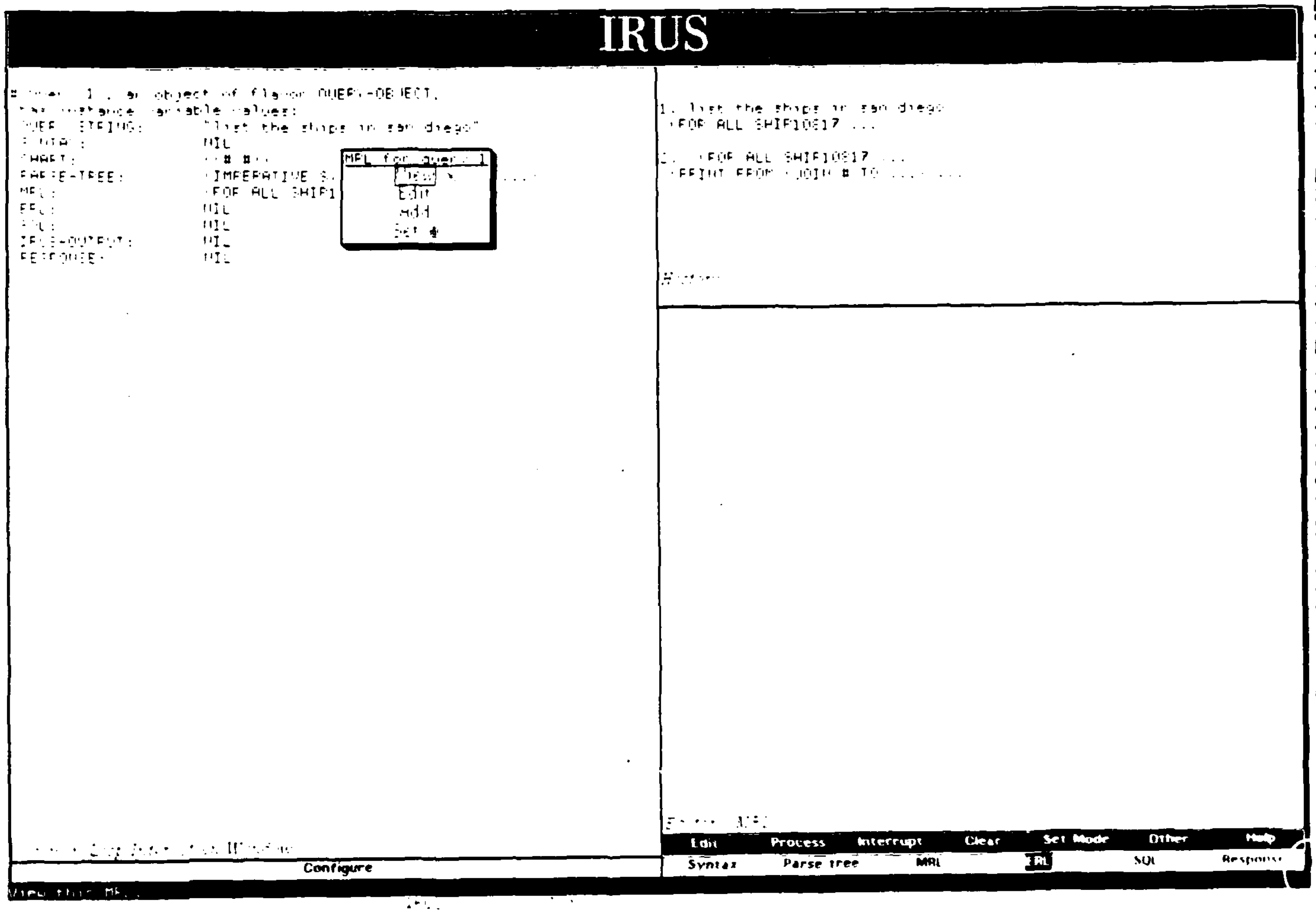


```
#<Query 1>, an object of flavor QUERY-OBJECT,
 has instance variable values:
  QUERY-STRING:        "list the ships in san diego"
  SYNTAX:              NIL
  WWFF:                ((# #))
  PARSE-TREE:          (IMPERATIVE SYNTAX (# # # ... ...
  MRL:                 (FOR ALL SHIP10517 ...)
  ERL:                 NIL
  SQL:                 NIL
  IRUS-OUTPUT:         NIL
  RESPONSE:            NIL
```

```
1. list the ships in san diego
 (FOR ALL SHIP10517 ...

2. (FOR ALL SHIP10517 ...
 (PRINT FROM (JOIN # T( ... ...
```

```
L:View this MRL, R:Menu of View, Edit, Add, Set-M.
                    IRU:
```

FIG. 37. SELECT MRL FROM INTERACTION PANE

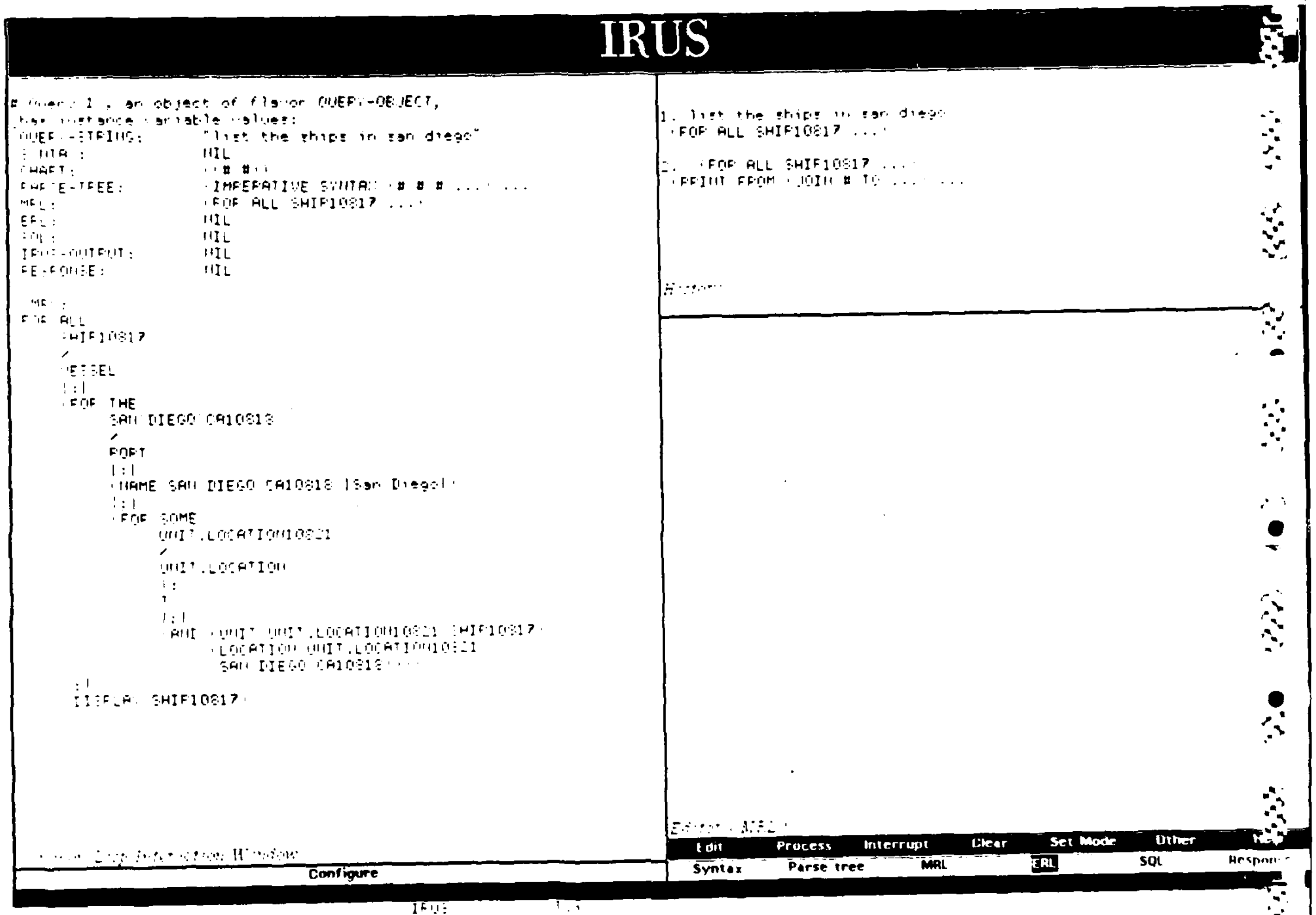


```
# query 1, an object of flavor QUERY-OBJECT,
has instance variable values:
 QUERY-STRING:          "list the ships in san diego"
 DATA:                  NIL
 CHART:                 #<...>
 PARSE-TREE:            (IMPERATIVE S...
 MRL:                   (FOR ALL SHIP1...
 IRL:                   NIL
 SQL:                   NIL
 IRUS-OUTPUT:           NIL
 RESPONSE:              NIL
```

```
1. list the ships in san diego
 (FOR ALL SHIP10817 ...

2.  (FOR ALL SHIP10817 ...
 (PRINT FROM (JOIN # TO ... ...
```

View this MRL

FIG. 38. SELECT VIEW COMMAND FOR MRL



```
#<Query 1>, an object of flavor QUERY-OBJECT,
 has instance variable values:
 QUERY-STRING:        "list the ships in san diego"
 SYNTAX:              NIL
 CHART:               (# #)
 PARSE-TREE:          (IMPERATIVE SYNTAX (# # # ...) ...)
 MRL:                 (FOR ALL SHIP10817 ...)
 ERL:                 NIL
 SQL:                 NIL
 IRUS-OUTPUT:         NIL
 RESPONSE:            NIL

 MRL:
 (FOR ALL
      SHIP10817
      /
      VESSEL
      |:|
      (FOR THE
           SAN DIEGO CA10818
           /
           PORT
           |:|
           (NAME SAN DIEGO CA10818 |San Diego|)
           |:|
           (FOR SOME
                UNIT.LOCATION10821
                /
                UNIT.LOCATION
                |:
                T
                |:|
                (AND (UNIT UNIT.LOCATION10821 SHIP10817)
                     (LOCATION UNIT.LOCATION10821
                      SAN DIEGO CA10818)))))
   :|
   (DISPLAY SHIP10817))
```

```
1. list the ships in san diego
 (FOR ALL SHIP10817 ...)

2.  (FOR ALL SHIP10817 ...)
 (PRINT FROM (JOIN # TO ...) ...)
```

FIG. 39. MRL DISPLAYED IN INTERACTION PANE

Finally, the developer is provided with configurations for editing the rules used for interpreting English sentences (the IRACQ configuration) and editing IRUS' dictionary of words (the IRUS Dictionary configuration). These configurations are selected using the **Configuration** command (see Figure 40).

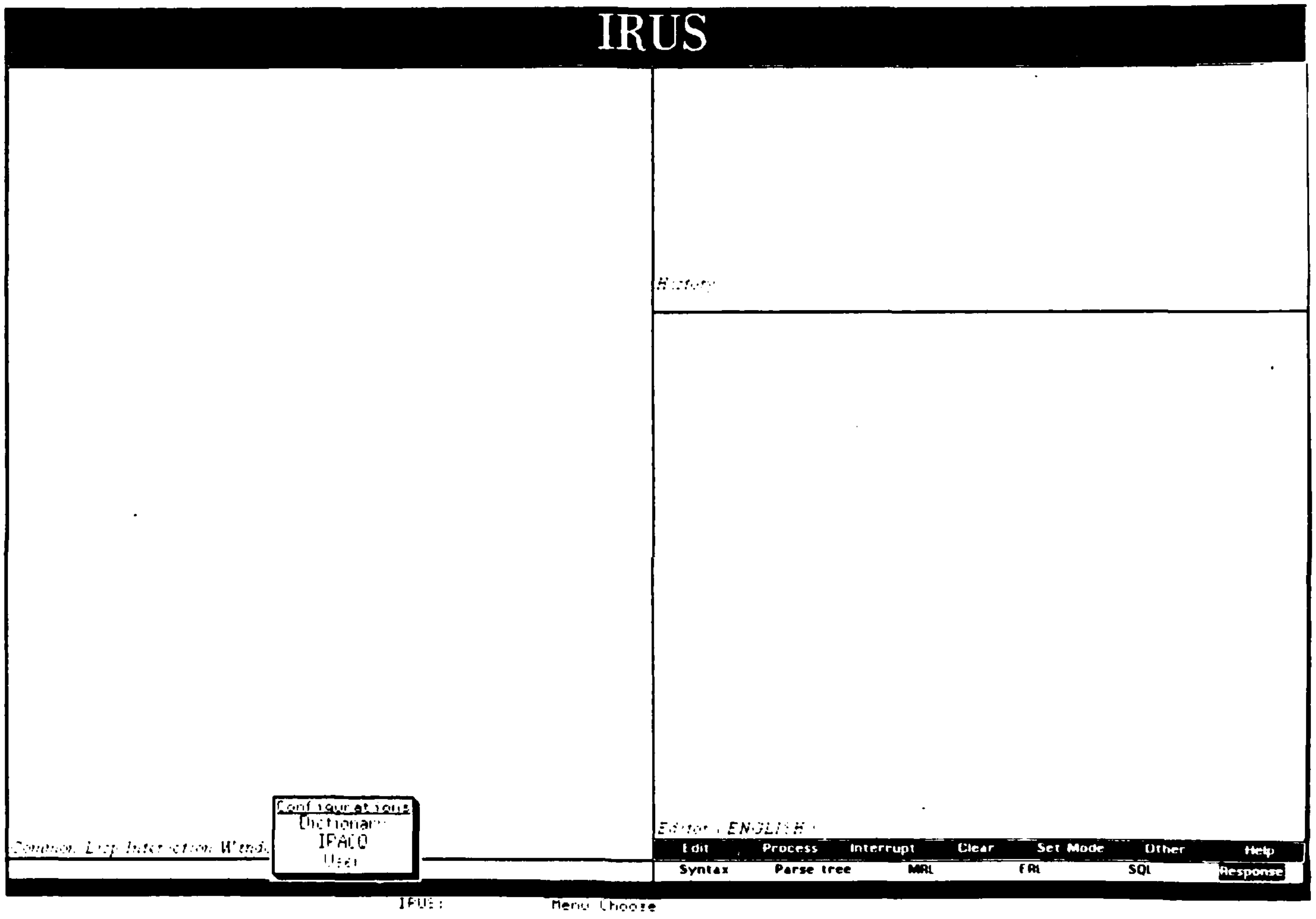


FIG. 40. CONFIGURATION MENU

For further detail on available commands and interaction with IRUS, see Appendices A and B.

6 Summary

The end-user's and developer's configurations of the IRUS interface illustrated here have several important features. A full editor for Lisp expressions or text is provided to aid in the composition or revision of input to IRUS; feedback on stages of processing is provided as processing proceeds; a history mechanism and facilities for incorporating help mechanisms or command composition aids is included; and the capability for setting breakpoints in IRUS processing and for examining and editing intermediate structures resulting from IRUS processing is available.

These facilities have been implemented in a flexible and modular interface system that can be adapted to a variety of applications and development needs.

APPENDIX A
IRUS INTERFACE DOCUMENTATION

1. To select an IRUS frame, hit <SELECT> R To create a new frame, hit <SELECT> CONTROL-R. (If no frame is built, call the function IRUS-I RESET and try again.)

2. To process a query (or set of queries), compose queries in the editor pane and click on PROCESS command or hit the <END> key (See next section for other available commands.) Queries are separated in the editor pane with two carriage returns (Actually, you can separate them by something else if you rebind the variable IRUS-I:*QUERY-DELIMITER*, which is currently bound to two carriage returns.)

3. To change configurations, click on the CONFIGURE command. Current configurations are: User, Developer, IRACQ, and Dictionary.

4. To get an IRUS pop-up frame, call the function IRUS-EDIT. (See description on page 41.)

5. Commands in editor command menu (developer configuration).

   o EDIT. get into the editor (equivalent to clicking the mouse in the editor pane.)

   o PROCESS: clicking left processes all the queries that are in the editor (equivalent to hitting the <END> key); clicking right pops up a menu allowing you to process a subset of the queries. A subset is designated by marking queries. Currently queries are marked by preceding them with an *.

   o INTERRUPT: get out of the editor pane and into the Lisp pane (equivalent to clicking in the Lisp pane).

   o CLEAR: clear the editor.

   o SET MODE: set the editor mode to ENGLISH, PARSE-TREE, MRL, ERL, or SQL. In ENGLISH mode, ZMACS auto-fill text mode is set. In the other modes, ZMACS Lisp mode is set.

   o OTHER. pops up a menu of other commands. WRITE, INSERT, ABBREV, and COMMENT.

      . WRITE. write queries that are in the editor to a file. You will be prompted for a file name.

      . INSERT. insert queries from a file into the editor. You will be prompted for a file name

      . ABBREV. pops up menu of abbreviation commands, e.g., turn on ZMACS word abbreviation mode, make new word abbreviation, list word abbreviations

. COMMENT: save a comment in the comments file. User is given the option of also saving query objects in the file.

- o HELP: pops up help menu. (This menu currently is under development.)

6. Developer's second command menu:

   - o This menu appears in the developer's configuration and allows incremental processing of queries. The query can be processed to SYNTAX (syntax only, no semantics), PARSE-TREE, MRL, ERL, SQL, or RESPONSE (from the database).

7. Commands in editor command menu (user configuration):

   - o The user's editor command menu contains a subset of the commands available for the developer: EDIT, PROCESS, INTERRUPT, CLEAR, OTHER, and HELP.

8. IRACQ interface:

   - o ADD IRULE: add an IRule. You will be asked several questions about the IRule to be added.
   - o EDIT IRULE: you will be prompted for an IRule name, and an editor pane containing the IRule will appear. An editor command pane will appear that contains the following commands: SAVE IRULE, INTERRUPT, ABORT, and HELP. (Note: when an IRule is edited, a query object is not created. The s-expression is put into the editor, and the IRule name is bound to IRUS:*IRULE-NAME*. The edited IRule is bound to IRUS:EDITED-IRULE.)
   - o PRINT IRULE: you will be prompted for an IRule name, and the IRule will be printed out on the Lisp pane.
   - o DELETE IRULE: you will be prompted for an IRule name, and the IRule will be deleted.
   - o SAVE IRULES: save all IRules that have been created or edited.

9. When a query is processed from the editor pane, a query object is created that contains instance variables for each of the stages in the processing. These instance variables are also objects. A query that starts from English, for example, contains a parse tree object whose value is the s-expression that is the result of parsing the English.

   The following commands are available for operating on query objects (i.e., are available from a menu that appears when you click on a query object):

   - o DESCRIBE: list the parts of the query object (in the Lisp pane).
   - o EDIT: edit this query object. (Anything already in the editor will be erased.)

- o ADD: add this query object to the editor. (Anything already in the editor will remain in the editor.) This command allows you to yank several queries into the editor to be processed at the same time.

- o REPEAT: repeat this query. Query is processed to the stage currently indicated by the developers' "incremental" command menu.

- o INSPECT: call the Lisp system function INSPECT on this object.

- o SET-@: set the value of the global variable IRUS:@ to this object.

10. The following commands are available for operating on objects that are parts of query objects:

- o VIEW: display the value of this object.

- o EDIT: edit this object. (Same as EDIT for query object.)

- o ADD: add this object to the editor. (Same as ADD for query object.)

- o SET-@ : (Same as for query object.)

11. Useful Functions

(Note: functions are in IRUS package unless otherwise specified.)

- o The following functions are available for getting the VALUES of the query parts (each takes a query object as an argument):

  QUERY-STRING, PARSE-TREE, MRL, ERL, SQL, RESPONSE, CHART, SYNTAX, IRUS-OUTPUT.

  These functions are available for getting the PARTS of the query:

  QUERY-STRING-OBJECT, PARSE-TREE-OBJECT, MRL-OBJECT, ERL-OBJECT, SQL-OBJECT, RESPONSE-OBJECT, CHART-OBJECT, SYNTAX-OBJECT, IRUS-OUTPUT-OBJECT.

- o IRUS-EDIT (&optional (make-new-frame nil)): *function*

  This function creates a pop-up IRUS frame that contains only an editor pane and an editor command menu. The size and location of the frame are specified by the variable IRUS-I:*DEFAULT-POPUP-EDGES*. (See description on page 44.) Currently, the command menu is the same as in the user configuration, so the incremental processing stages cannot be set. (The query will be processed from English to database response; currently the response is a dummy response.) An added feature appears in the OTHER command menu: a HISTORY option for creating a pop-up history frame. The size and location of the frame are specified by the variable IRUS-I:*DEFAULT-POPUP-HISTORY-EDGES*. (If a previous pop-up history frame exists, selecting HISTORY from the menu will display the previous frame rather than create a new one.) The pop-up IRUS frame and the pop-up history frame disappear when the INTERRUPT is selected from the editor command menu, and a list of all query objects processed is returned. Calling

IRUS-EDIT again selects the previous pop-up IRUS frame, or if called with an argument of T, creates and selects a new pop-up IRUS frame.

Either of these pop-up frames may be moved easily by positioning the mouse over the square in the lower right corner of the frame, holding down a mouse button, and dragging the mouse. The frames may be resized easily by positioning the mouse over the title line at the top of the frame, holding down a mouse button, and dragging the mouse.

o QDESCRIBE (object): *function*

This function partially describes an object; i.e., it only prints out relevant instance variables. It does not print out pointers to parent object, scroll-parse-item, and other things that probably are only important to someone working on the interface. (Calling DESCRIBE on the object will print out all the instance variables.) QDESCRIBE is what is called when you click on a query object in the history pane.

o QEDIT (thing): *function*

This function puts something into the editor pane. The "something" can be a query object, any of a query objects' parts (e.g. MRL object), a string, or any s-expression. If QEDIT is called with a query object or query-part object, the editor mode will be set automatically for you; otherwise you must set it.

o LAST-QUERY (&optional (frame IRUS-I:*FRAME*)): *function*

This function returns the most recently processed query object. The optional argument allows you to get the current query for the pop-up frame (which is bound to IRUS-I:*POPUP-FRAME*). (Note: the fullsize frame is always bound to IRUS-I:*TOPLEVEL-FRAME*; IRUS-I:*FRAME* is bound to the IRUS frame that is currently selected -- pop-up or fullsize.)

o PREVIOUS-QUERIES (&optional (frame IRUS-I:*FRAME*)): *function*

This function returns a list of the previously processed query objects. The optional argument allows you to get the current query for the pop-up frame (which is bound to IRUS-I:*POPUP-FRAME*). (Note: the fullsize frame is always bound to IRUS-I:*TOPLEVEL-FRAME*, IRUS-I:*FRAME* is bound to the IRUS frame that currently is selected -- pop-up or fullsize.)

o CURRENT-QUERY: *function*

This function returns the query object currently being processed.

o CURRENT-QUERIES: *function*

This function returns a list of the query objects currently being processed.

o QUERIES (&optional (frame IRUS-I:*FRAME*)). *function*

This function returns a list of all the queries that have been processed in the frame. The optional argument allows you to get the queries from the pop-up frame.

- GET-QUERY (n &optional (frame IRUS-I:*FRAME*)): *function*

  This functions returns the Nth query (assuming that query numbering begins with 1). The optional argument allows you to get the current query for the pop-up frame (which is bound to IRUS-I:*POPUP-FRAME*). (Note: the fullsize frame is always bound to IRUS-I:*TOPLEVEL-FRAME*; IRUS-I:*FRAME* is bound to the IRUS frame that is currently selected -- pop-up or fullsize.)

- IRUS-I:CLEAR-EDITOR: *function*

  Clears the editor.

- IRUS-I:GET-EDITOR-STRING (&key common): *function*

  Returns the string in the editor. With the keyword "common", returns a Common Lisp string.

- IRUS-I:SET-EDITOR-STRING (string): *function*

  Returns the string in the editor.

- IRUS-I:ADD-STRING-TO-EDITOR (string): *function*

  Adds string to editor at current cursor position.

- IRUS-I:ADD-TO-EDITOR (object): *function*

  Adds string associated with object to editor at current cursor position.

- IRUS-I:EXPOSE-HISTORY: *function*

  Pops up the history pane.

- IRUS-I:HIDE-HISTORY: *function*

  Causes the pop-up history pane to disappear.

- IRUS-I:ERASE-HISTORY: *function*

  Clears the history pane and its list of all queries that have been processed.

12. IRUS--DRIBBLE (filename): *function*

    Causes everything displayed on the Lisp pane also to be recorded in a file.

13. SET-FLAGS. *function*

    Pops up menu of query processing flags. clear editor pane after processing, clear Lisp pane before processing, ignore errors when processing, save debug output in dribble file when dribbling, deexpose pop-up frame after processing queries.

14 Global Variables

o IRUS-I:*FRAME*: *variable*

The currently selected IRUS frame (either pop-up or fullsize).

o IRUS-I:*TOPLEVEL-FRAME*, IRUS-I:*POPUP-FRAME*: *variable*

The fullsize IRUS frame is always bound to IRUS-I:*TOPLEVEL-FRAME*. The pop-up frame is always bound to IRUS-I:*POPUP-FRAME*.

o IRUS-I:*DEFAULT-POPUP-EDGES*: *variable*

This variable is the default screen location for the IRUS pop-up frame. It is a list of the left, top, right, and bottom pixel locations of the edges of a rectangle. Current default is '(45 450 510 650).

o IRUS-I:*DEFAULT-POPUP-HISTORY-EDGES*. *variable*

This variable is the default screen location for the IRUS pop-up history frame. It is a list of the left, top, right, and bottom pixel locations of the edges of a rectangle. Current default is '(590 300 850 680).

o IRUS-I:*PROMPT*: *variable*

The Lisp pane contains a prompt that you can change by modifying the variable IRUS-I:*PROMPT*.

o IRUS-I:*QUERY-FILE*, IRUS-I:*QUERY-DIRECTORY*: *variable*

These variables specify the default file name and directory for writing (inserting) queries from (into) the editor pane.

o

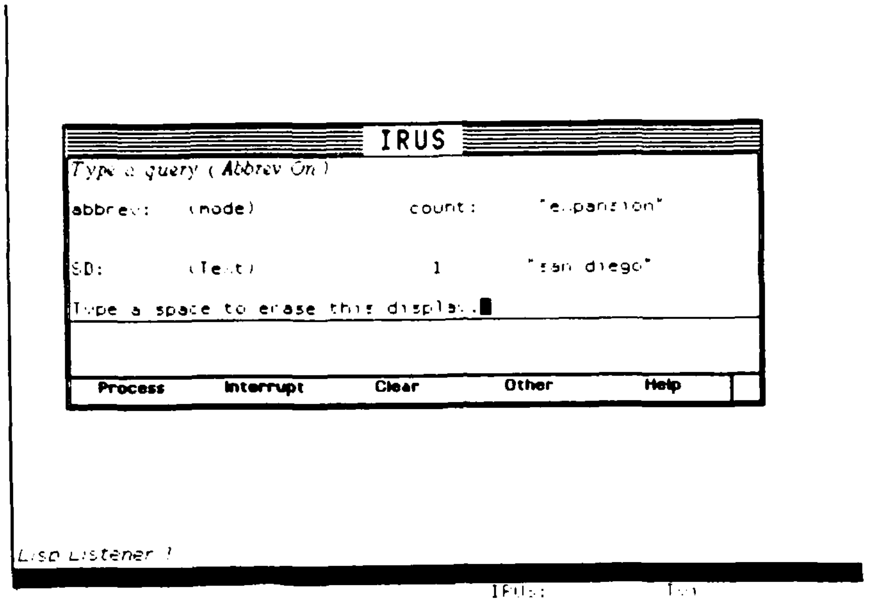
IRUS-I:*PRINT-STARTING-OBJECT*, IRUS-I:*PRINT-STARTING-OBJECT-NUMBER*, IRUS-I:*PRINT-ABBREV-STARTING-OBJECT*
variable These variables control what is printed on the Lisp pane after a query has been processed. (Note: They probably will become unnecessary as the system is used and a preferred printed representation becomes apparent.) The default is for the first two to be NIL and the last one to be T. In this case, a typical output will be:

MRL for "what is the Frederick's location and ...": <the MRL>.

The starting object, in this case the English sentence "what is the Frederick's location and readiness", is printed in abbreviated form; i.e., only as much of it as will fit on one line is printed. To have the entire sentence printed out, set IRUS-I:*PRINT-STARTING-OBJECT* to T. To have the query number printed out (e.g. MRL for Query 2), set IRUS-I:*PRINT-STARTING-OBJECT-NUMBER* to T. (Note: The printing function examines the variables in the order listed above, so if more than one is T, the first one in the ordering will determine what is printed.)

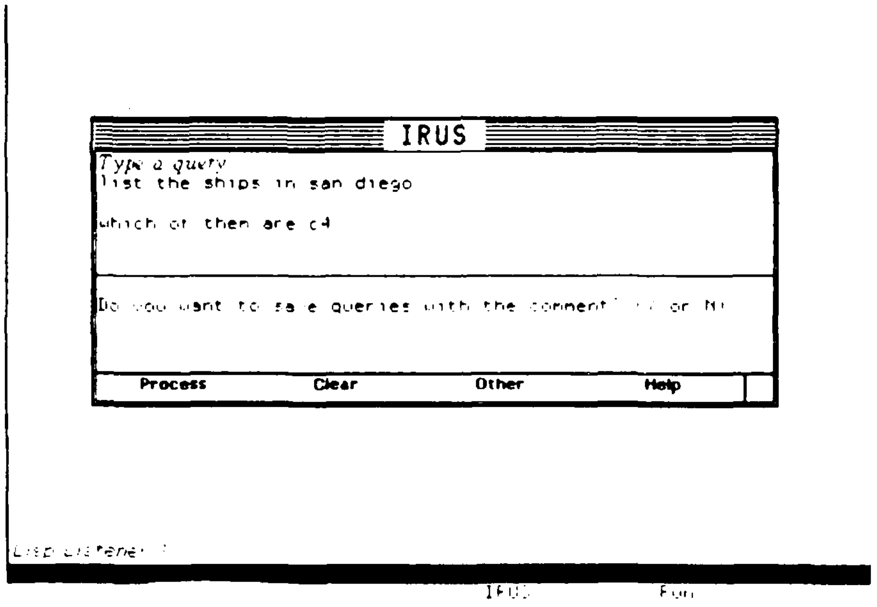
APPENDIX B
MORE SAMPLE SCREENS

This appendix contains sample screens showing the following capabilities and configurations: listing all word abbreviations, saving queries and comments, moving and reshaping the pop-up windows, editing an English interpretation rule, the IRUS Dictionary configuration, and an example of a full screen end-user configuration.



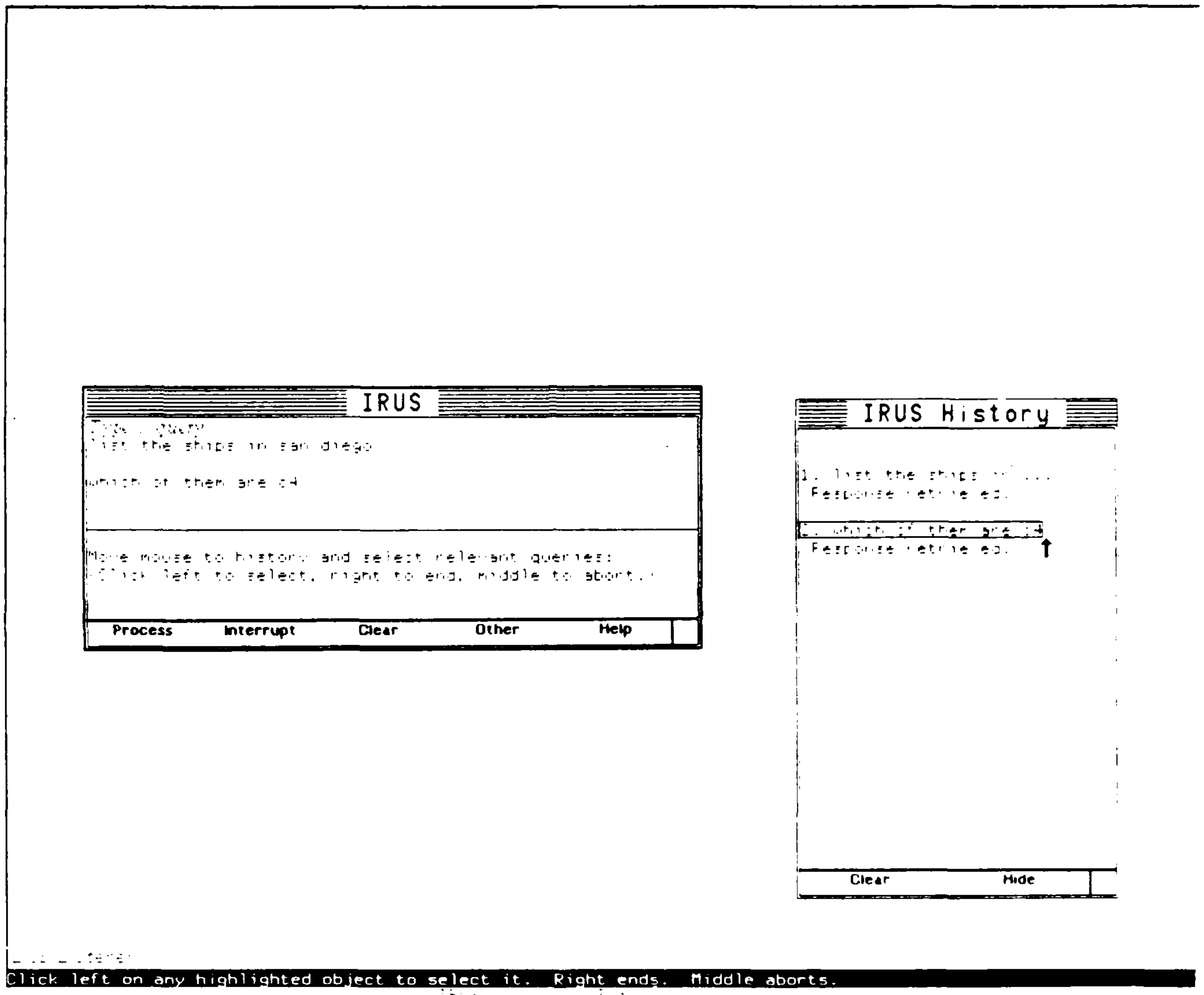
```
Type a query ( Abbrev On )
abbrev:   (node)              count:    "expansion"

SD:       (Text)                1       "san diego"

Type a space to erase this display.
```

FIG. 41   LIST ALL WORD ABBREVIATIONS



```
Type a query
list the ships in san diego

which of them are c4

Do you want to save queries with the comment? Y or N
```

FIG. 42   PROMPT FOR SAVING QUERIES IN COMMENT

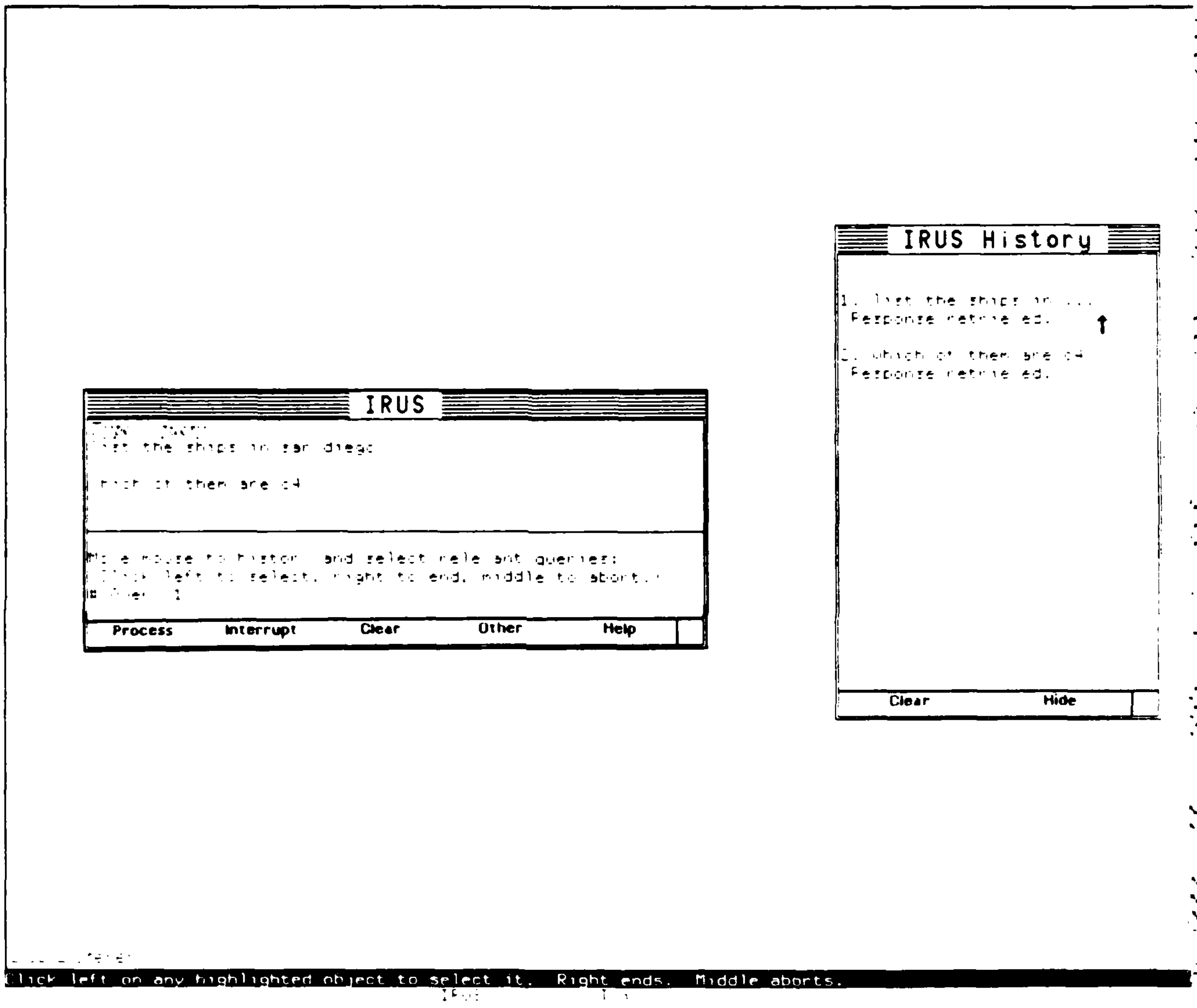


```
Type a query
list the ships in san diego

which of them are c4

Move mouse to history and select relevant queries:
(Click left to select, right to end, middle to abort.)
```

```
1. list the ships in ...
 Response retrieved.

2. which of them are c4
 Response retrieved.
```

```
Click left on any highlighted object to select it.  Right ends.  Middle aborts.
```

FIG 43. PROMPT FOR SELECTING QUERIES TO BE SAVED

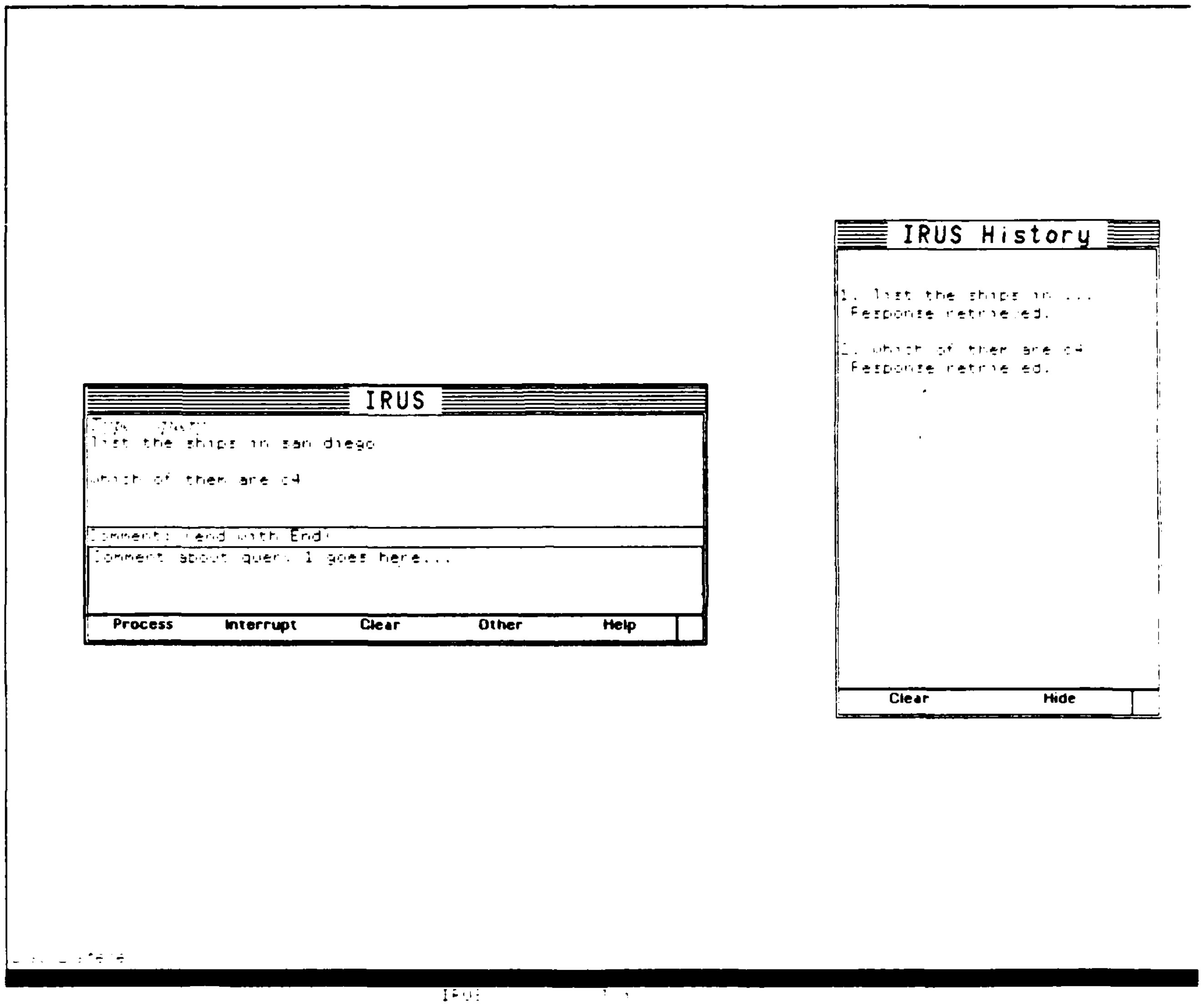


```
list the ships in san diego

which of them are c4

Move mouse to history and select relevant queries:
(Click left to select, right to end, middle to abort.)
[illegible]
```

```
1. list the ships in ...
 Response retrieved.

2. which of them are c4
 Response retrieved.
```

```
Click left on any highlighted object to select it.  Right ends.  Middle aborts.
```

FIG 44   QUERY SELECTED TO BE SAVED

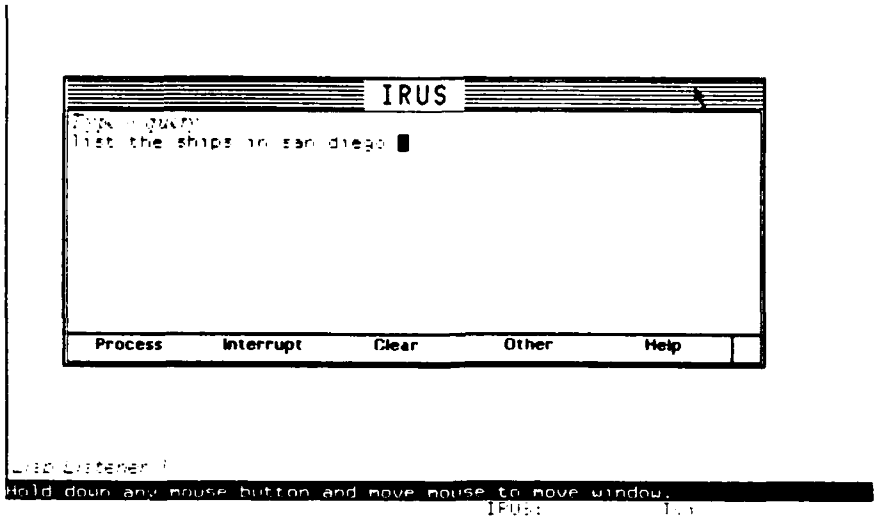


```
list the ships in san diego

which of them are c4
```

```
Comment: (end with End)
Comment about query 1 goes here...
```

```
1. list the ships in ...
 Response retrieved.

2. which of them are c4
 Response retrieved.
```

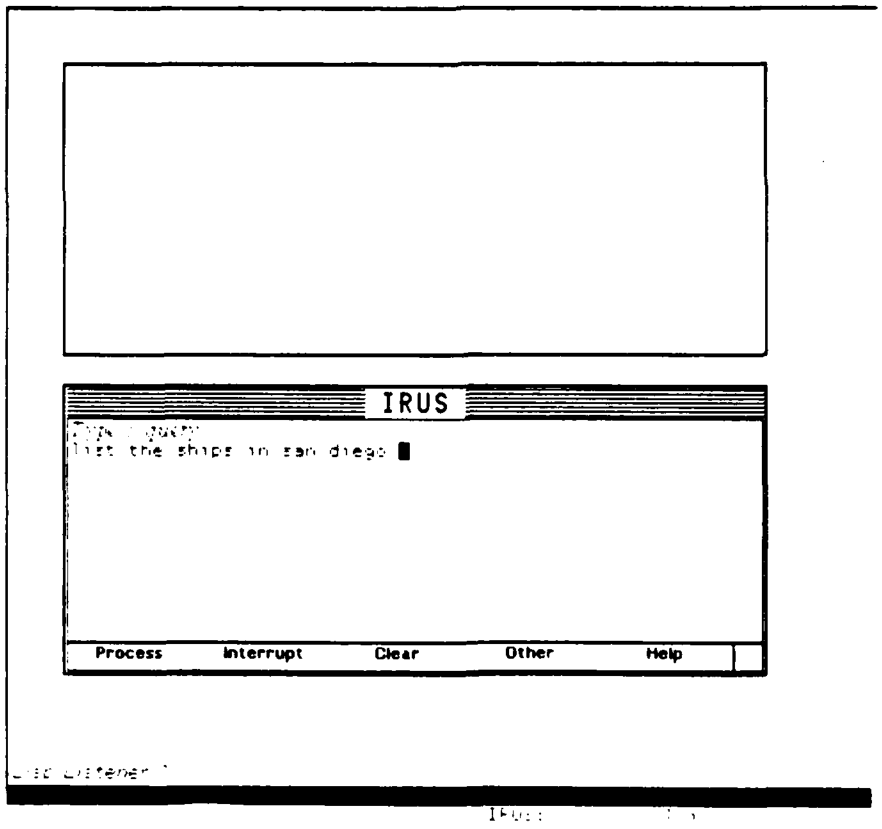
FIG 45 ENTER COMMENT (CLICKED RIGHT TO END QUERY SELECTION)



```
Type a query
list the ships in san diego
```

```
Lisp Listener 1
Hold down any mouse button and move mouse to move window.
IRUS:
```

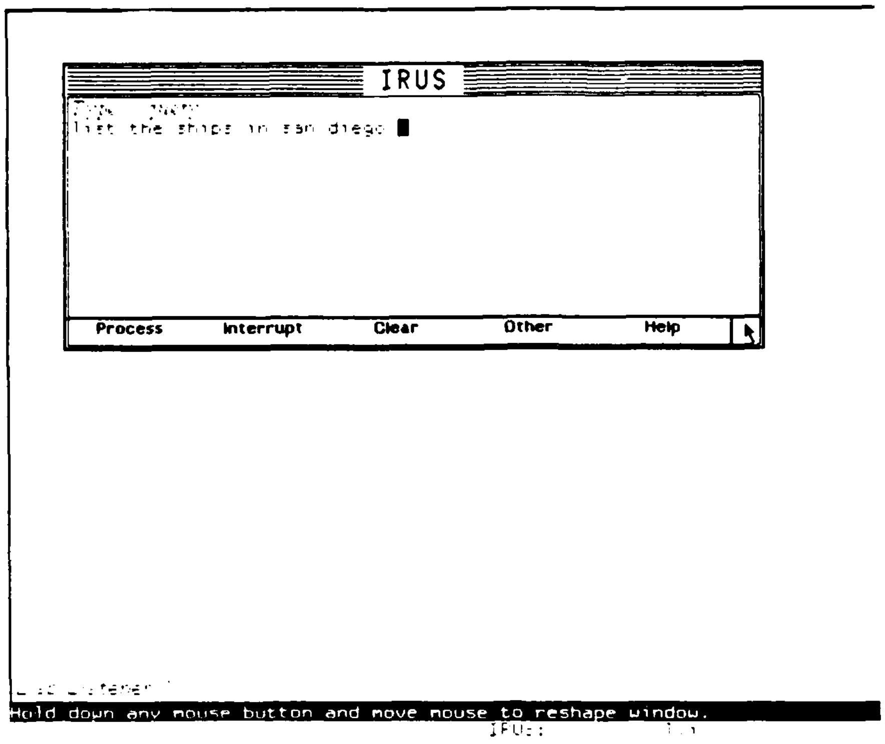
FIG. 46. MOVE MOUSE TO TOP OF WINDOW



```
Type a query
list the ships in san diego
```

```
Lisp Listener 1
IRUS:
```

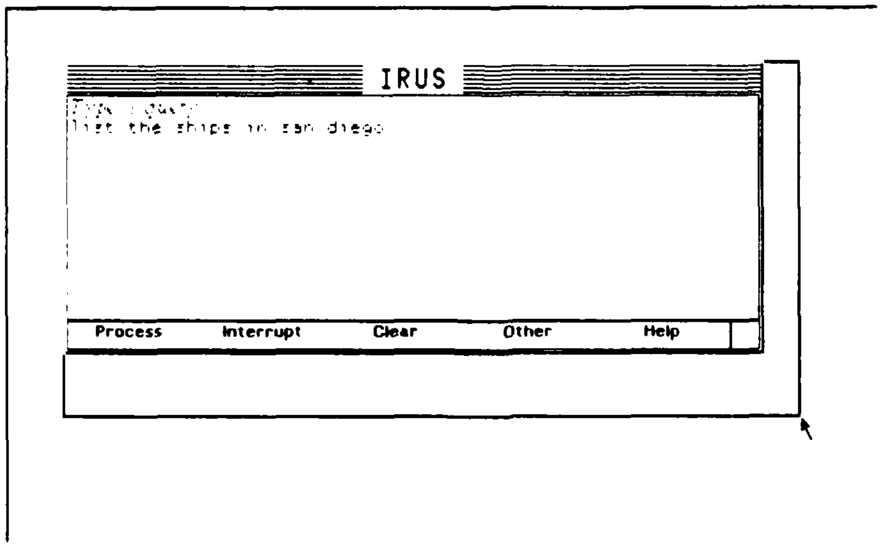
FIG. 47. HOLD DOWN BUTTON AND MOVE MOUSE



```
[illegible]
list the ships in san diego
```

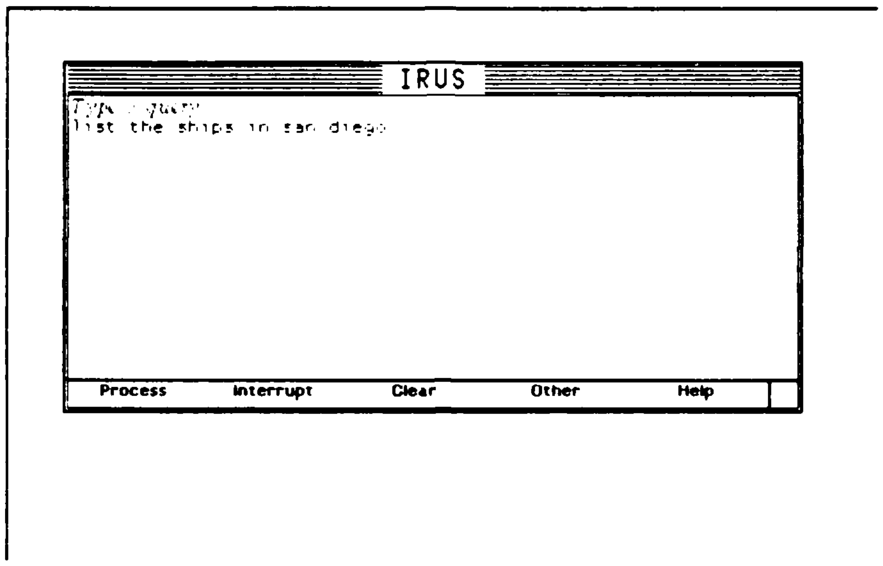
```
[illegible]
Hold down any mouse button and move mouse to reshape window.
IFU:
```

FIG. 48. RELEASE BUTTON FOR NEW POSITION, MOVE MOUSE TO LOWER RIGHT CORNER



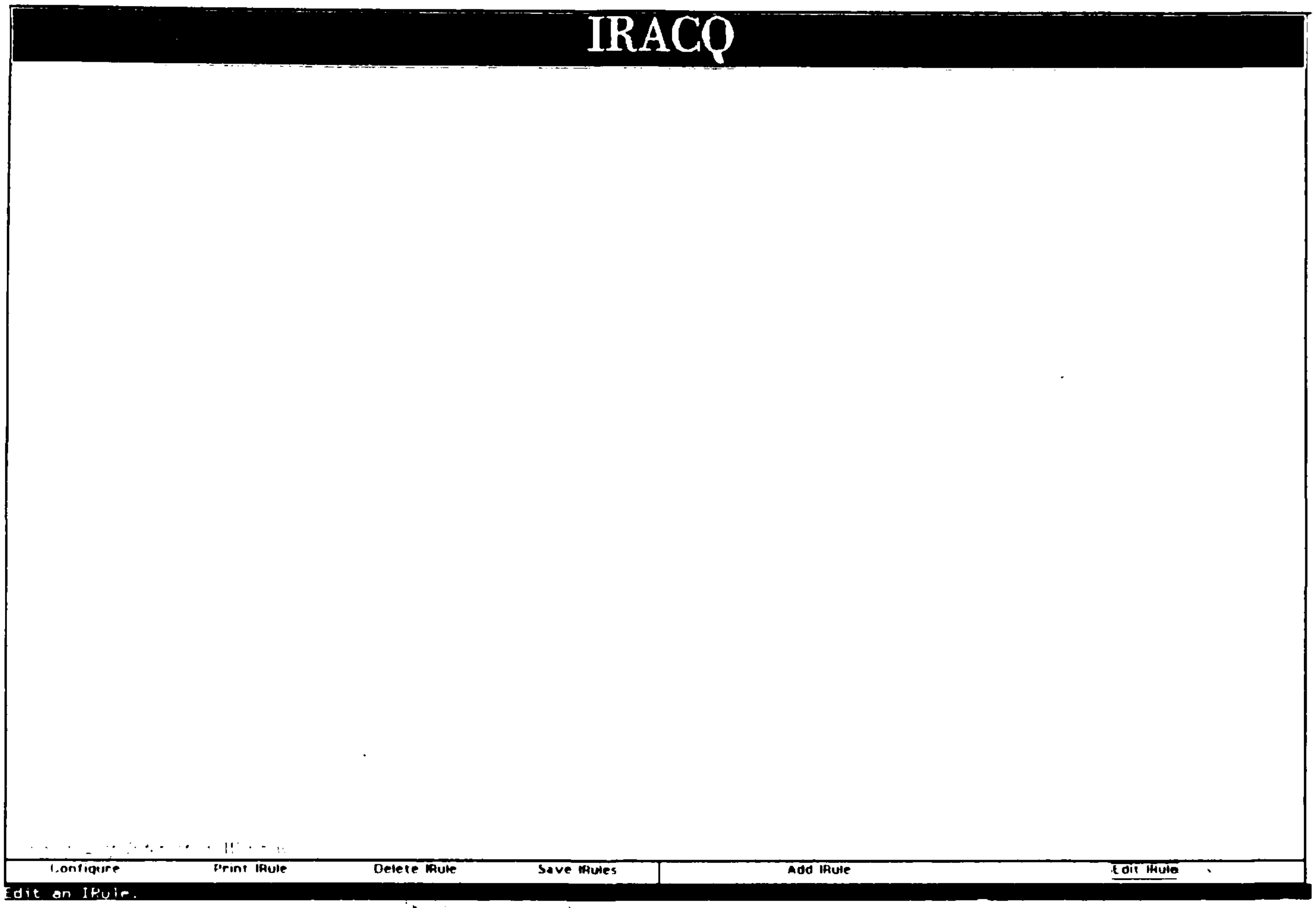
```
[illegible]
list the ships in san diego
```

FIG. 49. HOLD DOWN BUTTON AND MOVE MOUSE



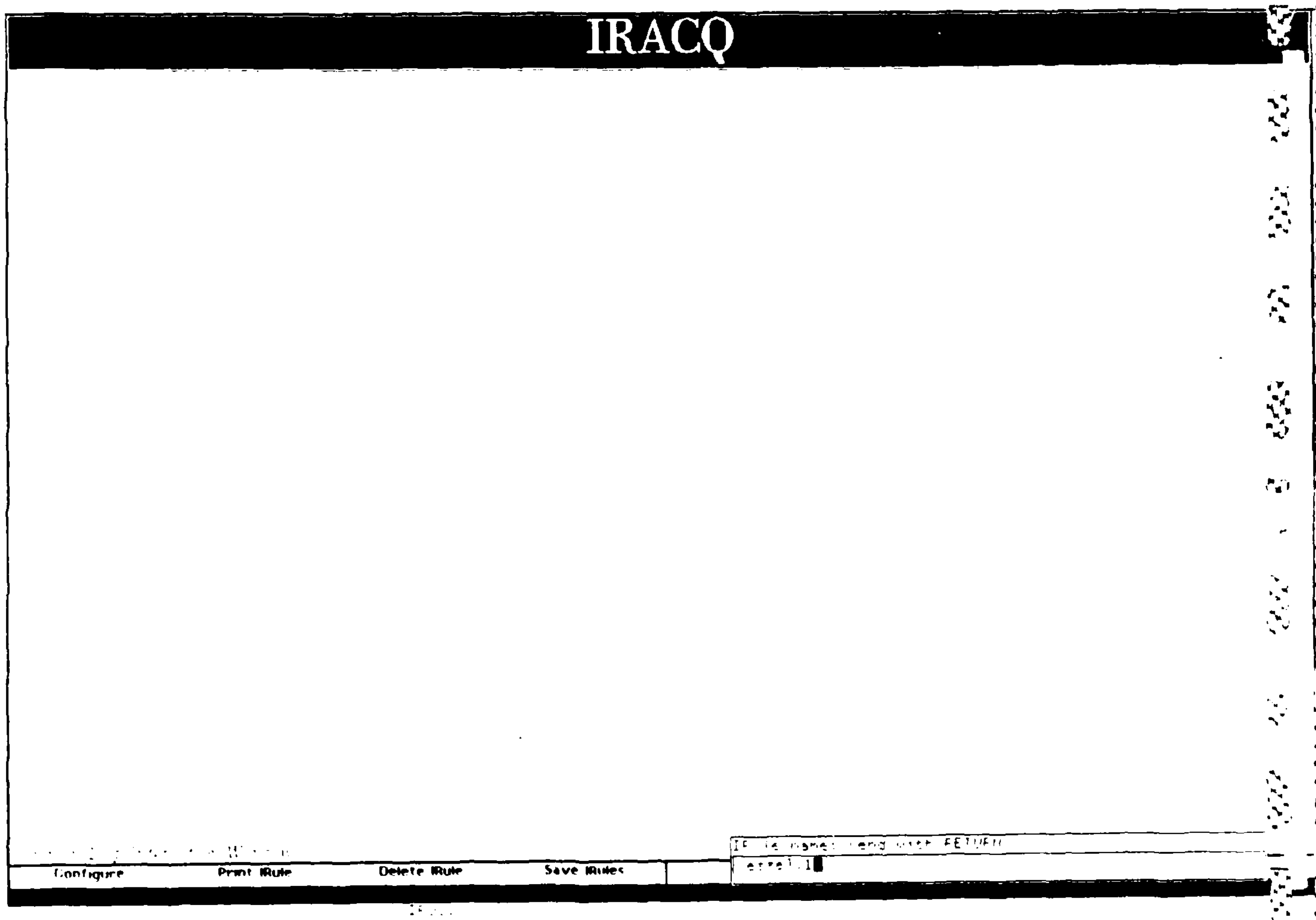
```
Type a query
list the ships in san diego
```

FIG. 50. RELEASE MOUSE BUTTON FOR NEW POSITION



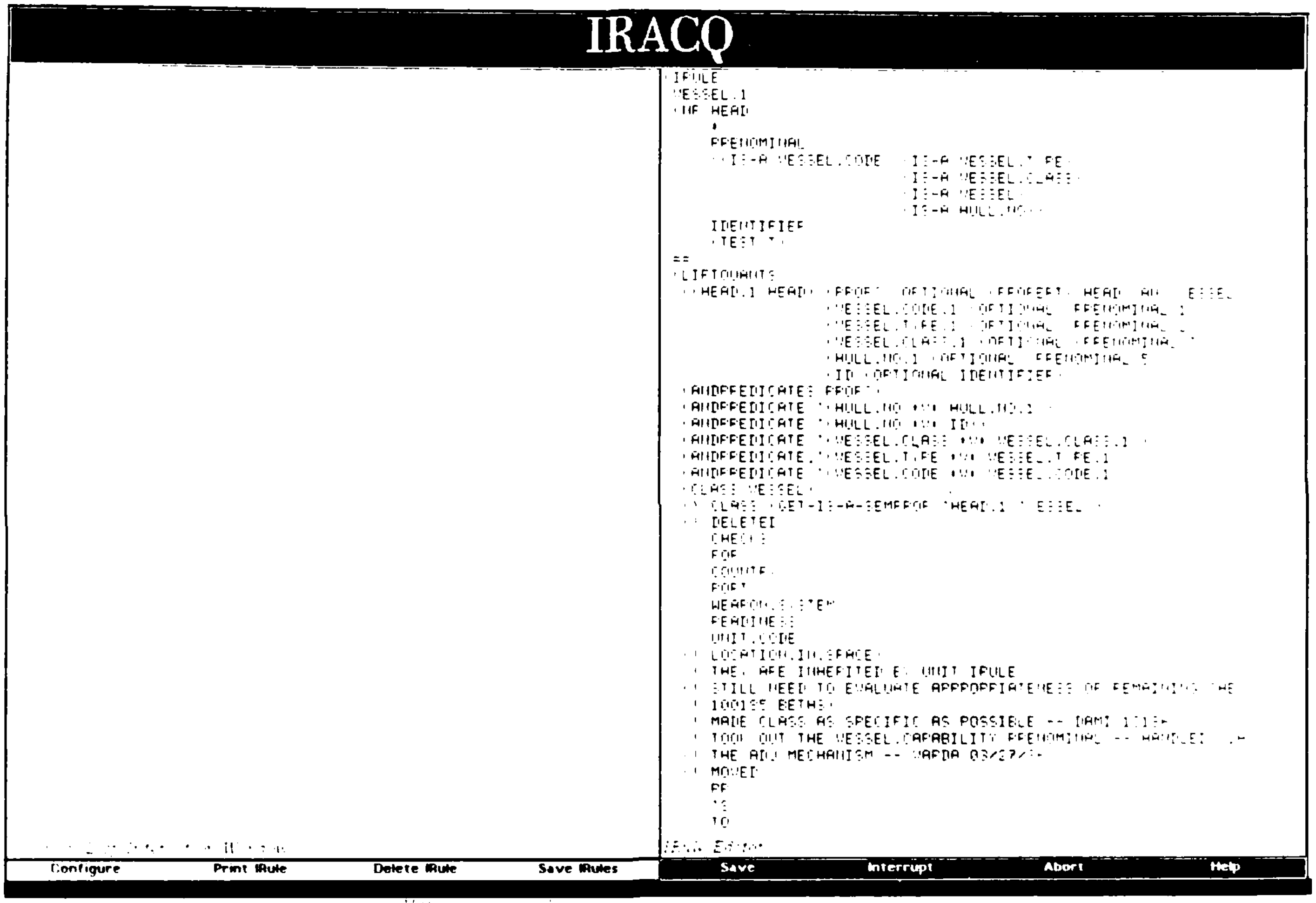
Edit an IRule.

FIG. 51. IRACQ CONFIGURATION, SELECT EDIT IRULE COMMAND



IF le name: (end with RETURN)
errel.1

FIG 52   ENTER IRULE NAME



```
(IRULE
VESSEL.1
(NP HEAD
    *
    PRENOMINAL
    ((IS-A VESSEL.CODE  (IS-A VESSEL.TYPE)
                        (IS-A VESSEL.CLASS)
                        (IS-A VESSEL)
                        (IS-A HULL.NO)))
    IDENTIFIER
    (TEST T))
==
(LIFTQUANTS
 ((HEAD.1 HEAD) (PROP (OPTIONAL (PROPERTY HEAD AN VESSEL
                  (VESSEL.CODE.1 (OPTIONAL PRENOMINAL 1
                  (VESSEL.TYPE.1 (OPTIONAL PRENOMINAL 2
                  (VESSEL.CLASS.1 (OPTIONAL (PRENOMINAL 3
                  (HULL.NO.1 (OPTIONAL PRENOMINAL 5
                  (ID (OPTIONAL IDENTIFIER)
 (ANDPREDICATES PROP)
 (ANDPREDICATE '(HULL.NO *V* HULL.NO.1 )
 (ANDPREDICATE '(HULL.NO *V* ID))
 (ANDPREDICATE '(VESSEL.CLASS *V* VESSEL.CLASS.1 )
 (ANDPREDICATE.'(VESSEL.TYPE *V* VESSEL.TYPE.1
 (ANDPREDICATE '(VESSEL.CODE *V* VESSEL.CODE.1
 (CLASS VESSEL)
 (* CLASS (GET-IS-A-SEMPROP 'HEAD.1 'VESSEL )
 (* DELETED
    CHECKS
    FOR
    COUNTRY
    PORT
    WEAPON.SYSTEM
    READINESS
    UNIT.CODE
 (* LOCATION.IN.SPACE)
 (* THEY ARE INHERITED BY UNIT IRULE
 (* STILL NEED TO EVALUATE APPROPRIATENESS OF REMAINING THE
 (* 100195 BETHS)
 (* MADE CLASS AS SPECIFIC AS POSSIBLE -- DAMI 1113)
 (* TOOK OUT THE VESSEL.CAPABILITY PRENOMINAL -- HANDLED BY
 (* THE ADJ MECHANISM -- VARDA 03/27/8)
 (* MOVED
    PR
    IS
    TO
```

FIG 53. IRULE APPEARS IN EDITING PANE

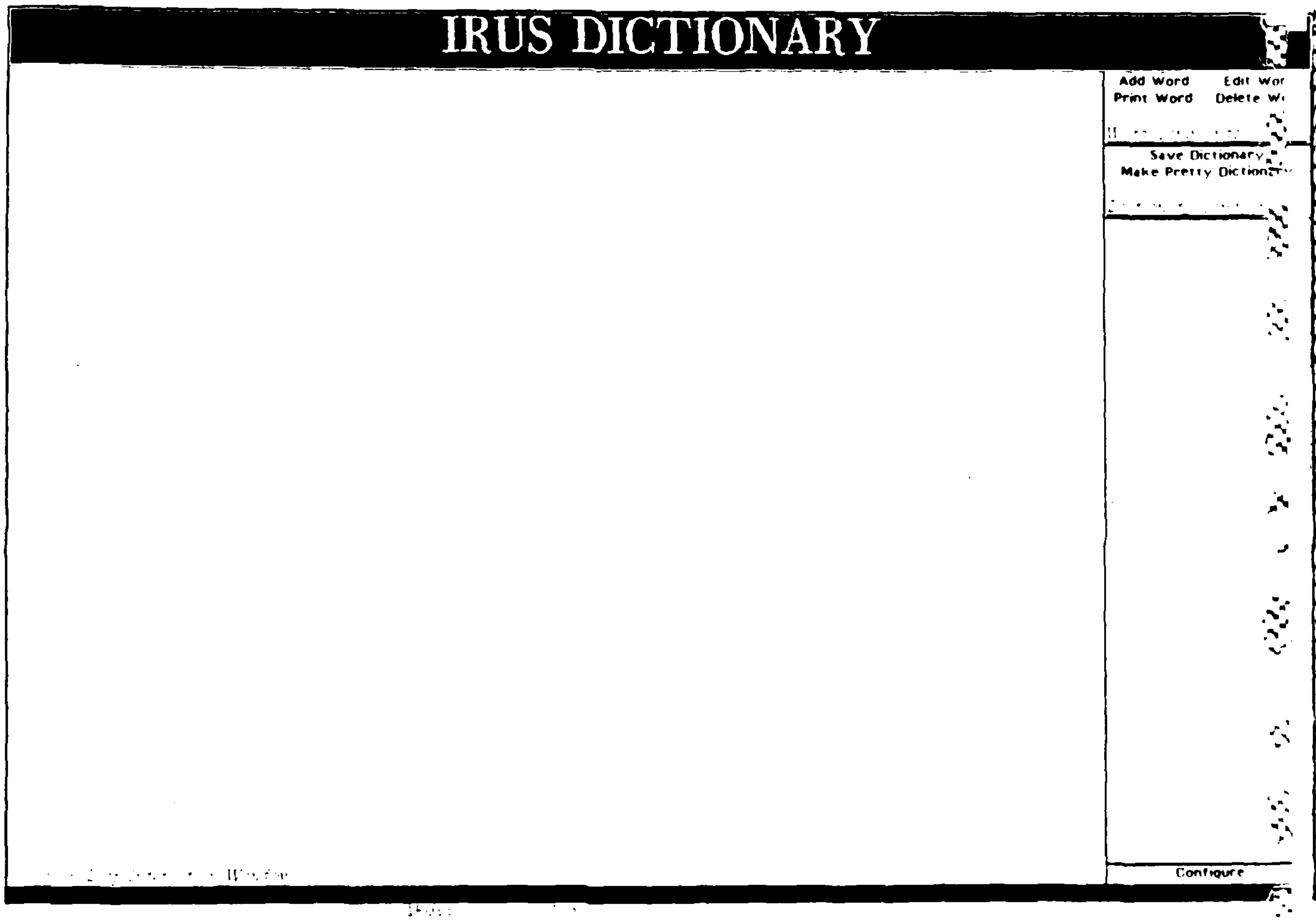


FIG 54   DICTIONARY CONFIGURATION

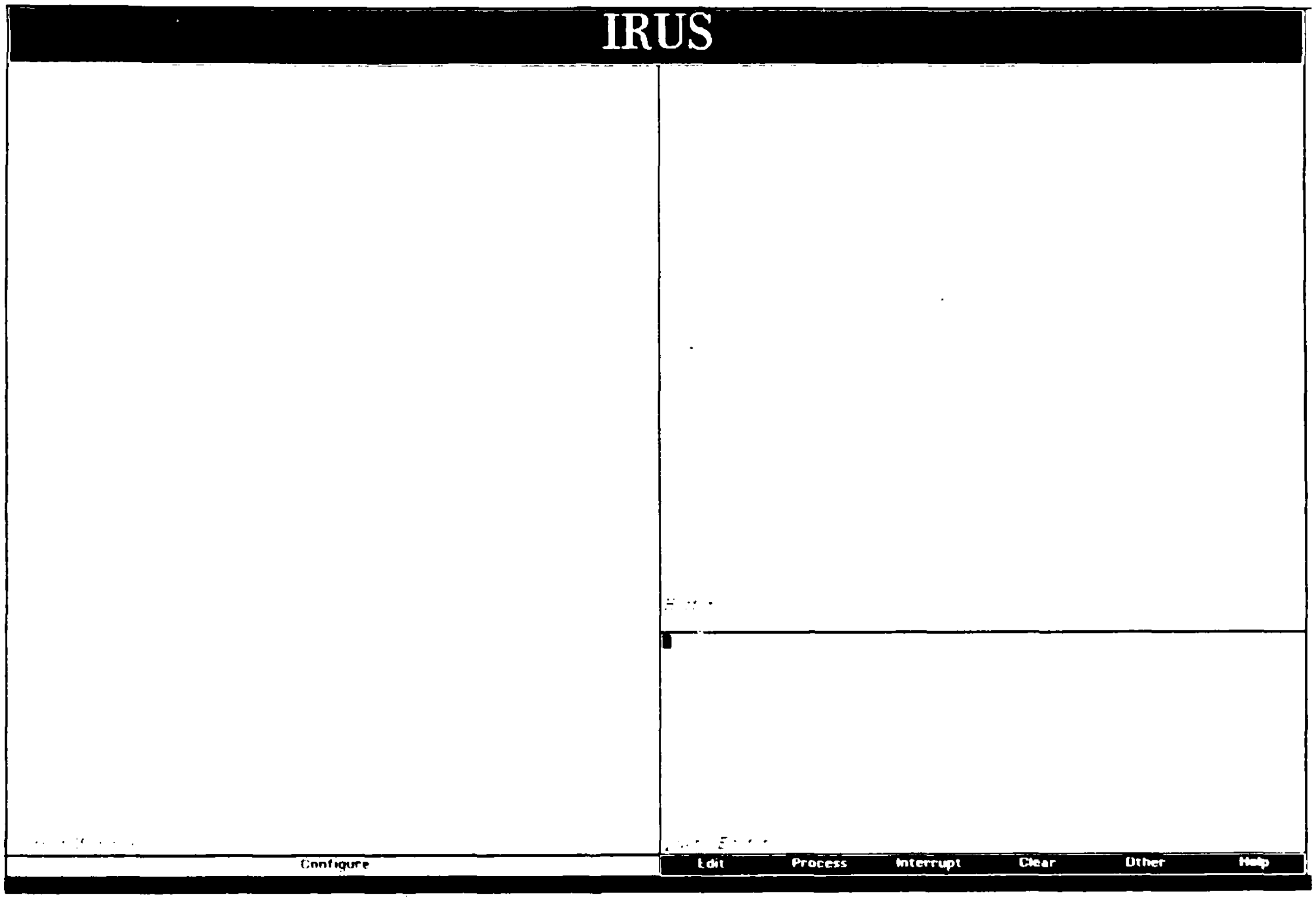


FIG 55 FULL SCREEN END-USER CONFIGURATION

BIBLIOGRAPHY

[1] Weischedel, R., *et al.*.
Out of the Laboratory: A Case Study with the IRUS Natural Language Interface.
In preparation.

[2] Bates, M., Stallard, D., Moser, M.
The IRUS Transportable Natural Language Database Interface.
Expert Database Systems.
Cummings Publishing Company, Menlo Park, CA, 1985.

[3] Shapiro, R.
WQL -- Window Query Language.
In preparation.